Web-käyttöliittymien automaattitestauksen menetelmät ja työkalut

Joonas Magnússon

Helsinki
Pro gradu -tutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet – Faculty	Laitos – Institution – Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä – Författare – Author

Joonas Magnússon

Työn nimi – Arbetets titel – Title

Web-käyttöliittymien automaattitestauksen menetelmät ja työkalut

Oppiaine – Läroämne – Subject

Tietojenkäsittelytiede

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Pro gradu	30.10.2019	54 sivua

Tiivistelmä – Referat – Abstract

Tässä tutkielmassa vertaillaan vertaillaan keskenään web-sovellusten käyttöliittymien automaattitestauksen menetelmiä. Tutkielmassa selvitetään mitä erityistä web-ohjelmistojen testaamisessa on, ja miten niitä on mahdollista testata. Tutkielmassa vertaillaan myös muutamia testiautomaation tekniikoita ja niitä hyödyntäviä ohjelmointikirjastoja ja työkaluja. Cypress on Javascript-pohjainen testiautomaatio kehys web-ohjelmistojen testaamiseen. Selenium Webdriver on Wedriver protokollaa hyödyntävä selainautomaatiokirjasto, jota käytetään mm. selainohjelmistojen testaamiseen. Sikuli on automaatiokirjasto, joka perustuu kuvantunnistukseen. Näitä kirjastoja ja niiden soveltamia tekniikoita vertaillaan keskenään niiden web-ohjelmistojen testaamiseen soveltuvuuden, testien kehityksen ja suorituksen tehokkuuden sekä alusta- ja selaintuen näkökulmista.

ACM Computing Classification System (CCS):
Software testing and debugging,
Web applications

Avainsanat – Nyckelord ⏍ Keywords

web-ohjelmistot, automaattitestaus

Säilytyspaikka ⏍ Förvaringställe ⏍ Where deposited

Muita tietoja ⏍ Övriga uppgifter ⏍ Additional information

Sisällysluettelo

1 Johdanto

Web-sovellukset ovat tärkeä osa jokapäiväistä elämää. Web-sovellusten avulla voi ostaa mitä tahansa. Web-sovellusten kautta varataan aikoja terveydenhuollon palveluihin. Web-sovelluksilla pidetään yhteyttä muihin ihmisiin, niiden kautta kulutetaan viihdettä kuten musiikkia ja elokuvia, web-sovelluksilla viestitään viranomaisten kanssa. Web-ohjelmistot ovat erottamaton ja lähes kriittinen osa miljoonien ihmisten arkipäiväistä toimintaa. Niiden kautta liikkuu rahaa ja yksityistä tietoa. On selvää, että tällaisten ohjelmistojen tulee toimia siten kuten niiden on tarkoitettu toimivan ja niiden laadusta tulee pitää hyvää huolta. Ohjelmistotestaus on tärkeä työkalu ohjelmistojen oikeanlaisen toiminnan varmistamisessa. Testauksen avulla voidaan selvittää toimiiko ohjelmisto tarkoituksensa mukaisesti ja testauksen avullaan voidaan havaita ohjelmiston virheitä niiden korjaamisen mahdollistamista varten.

Nykyaikaiset web-ohjelmistot ovat monimutkaisia ohjelmistoja. Web-ohjelmiston käyttöliittymä suoritetaan verkkoselaimessa, josta sovellus tekee internetin yli kutsuja yhteen tai useampaan taustapalveluun. Perinteisissä websovelluksissa selaimen avulla tehdään HTTP-kutsu web-palvelimelle, joka palauttaa vastauksena HTML-dokumentin selaimelle. Selain luo HTML-dokumentin pohjalta käyttäjälle ihmisluettavan verkkosivun, joka voi sisältää tekstiä, erilaista multimediaa sekä linkkejä osoitteisiin, joihin tehdä uusia HTTP-kutsuja ja saada vastauksena uusia resursseja, kuten HTML-dokumenteja. Selaimen web-palvelimille tekemiä kutsuja voidaan myös suorittaa asynkronisesti käyttöliittymän taustalla ja kutsujen vastausten pohjalta käyttöliittymän tilaa voidaan muokata lennosta javascriptin ja selainten toteuttaman DOM-rajapinnan avulla. Modernit web-ohjelmistot ovat rinnakkaisuutta sisältäviä rikkaan käyttöliittymän omaavia hajautettuja ohjelmistoja.

Ohjelmiston käyttöliittymän automaattitestaamisella tarkoitetaan testien suorituksen ja tulosten tarkistuksen automatisointia, jossa pyritään käyttämään testattavaa ohjelmistoa ihmiskäyttäjälle tarkoitetun käyttöliittymän kautta. Käyttäliittymän ohjaaminen ohjelmiston avulla tapahtuu tunnistamalla testattavasta käyttöliittymästä ohjelmiston suoritusaikana käyttöliittymän elementtejä ja tutkimalla näiden elementtien tilaa sekä antamalla erilaisille interaktiivisille elementeille syötteitä ja käskyjä käyttäen erilaisia tekniikoita. Käyttöliittymän komponentteja web-sovelluksessa voidaan tunnistaa muutamilla eri tekniikoilla. Pääasialliset tekniikat ovat verkkosivun elementtien erilaisiin tunnisteisiin perustuva elementtien tunnistus ja verkkosivun elementtien visuaalinen tunnistus, eli verkkosivun elementtien vertaaminen ennalta tallennettuihin kuvankaappauksiin.

Käyttöliittymävetoinen automaattitestaus mahdollistaa ohjelmiston täysimittaisen toistuvan toiminnallisen testaamisen. Käyttöliittymävetoisessa testaamisessa koko ohjelmisto kaikkine osineen on mahdollista testata todellisessa suoritusympäristössään. Tällöin saadaan havaittua sellaisia ongelmia, joita ei suppeammilla tai pienempiin testattavan ohjelmiston osiin keskittyvillä testausmenetelmillä välttämättä saada havaittua. Koska koko ohjelmiston koko läpimitan käsittävä käyttöliittymästä taustapalveluihin ja takaisin palaavan toiminnan testaaminen on hidasta suorittaa, myös suppeammalla laajuudella testaaminen on tarpeellista.

Tutkimuskysemys 1: Millaisia automaatiotekniikoita on hyödyllistä käyttää web-ohjelmistojen käyttäliittymävetoiseen testaamiseen? Työn tarkoituksena on kartoittaa eri lähestymistapojen vahvuuksia ja heikkouksia sekä selvittää miten erilaisia tekniikoita voidaan käyttää yhdessä. Työssä käyn läpi tutkimusta web-sovellusten käyttöliittymätestauksesta, sekä käyttöliittymätestauksen menetelmistä ja työkaluista. Kerään myös yhteen tutkimuksia menetelmien sekä työkalujen vertailusta. Pohjustan aihetta selvittämällä mitä testaaminen on, sekä miksi ja miten ohjelmistoja testataan. Käyn läpi testaamisen perusteita ja

käsitteitä. Esittelen testaamisen eri tasot ohjelmistotuotannon näkökulmasta, sekä eri tapoja testien suunnittelemiseen.

Tutkimuskysymys 2: Miten eri automaatiotekniikat vertautuvat toisiinsa web-ohjelmistojen käyttäliittymävetoiseen testaamiseen soveltuvuuden, testien luomisen ja kehittämisen, testien suoritustehokkuuden, sekä testien alusta ja -selaintuen saralla? Keskityn DOM-pohjaisten ja kuvantunnistukseen perustuvien menetelmien ja työkalujen vertailuun. Lupaavia testityökaluja ovat Selenium, Cypress ja Sikuli. Selenium Webdriver on selainautomaatiokirjasto, jossa on DOM-pohjainen käyttöliittymäkomponenttien tunnistus. Myös Cypress on DOM-pohjainen web-sovellusten testausohjelmointikehys, mutta se lähestyy web-sovellusten automaatiota eri tavalla kuin Selenium. Sikuli on kuvantunnistukseen perustuva käyttöliittymien automatisointikirjasto. DOM-pohjaisista automatisointityökaluista vertailen vielä keskenään verkkosivulle injektoitavaan javascriptiin pohjautuvaa automatisointia ja selainten tarjoamiin automaatiotyökaluihin perustuvaa automatiosointia. Vertailu perustuu aiempien tutkimusten tuloksiin sekä vertailtaviksi valittujen testaus- ja automaatiotyökalujen dokumentaatioon.

Tutkielman luvussa 2 esittelen lyhyesti testaamisen perusteita. Luvussa 3 esittelen web-ohjelmistojen toimintaa ja luvussa 4 selvitän mitä erityistä niiden testaamisessa on. Esittelen muutamia yleisiä web-ohjelmistojen testaamiseen soveltuvia automaatiotekniikoita ja tekniikoista hyödyntäviä työkaluja ja kirjastoja. Luvussa 5 käsitellään web-ohjelmistojen automaatio- ja testaustyökalujen sekä -menetelmien tärkeitä ominaisuuksia, sekä vertaan eri työkaluja ja menetelmiä suhteessa valittuihin ominaisuuksiin.

2 Ohjelmistojen testaus

Tässä luvussa käydään läpi ohjelmistotestauksen tarkoitus, ohjelmistotestauksen käsitteistöä, sekä ohjelmistotestauksen tapoja ja metodiikkaa. Luvun tarkoituksena on kertoa mitä ohjelmistotestaus on, miksi ohjelmistoja testataan ja miten ohjelmistojen testaaminen tapahtuu. Aliluvussa 2.1 käydään läpi hieman testaamisen peruskäsitteistöä. Aliluvussa 2.2 käsitellään pintapuoliseste testauksen tasoja.

Testaus on ohjelmistokehityksen laadunvalvonnan väline. Testauksessa testattavaa ohjelmistoa tai sen osaa suoritetaan jollain tietyllä syötteellä tai tavalla ja sen tulostetta tarkastellaan. Testauksen tarkoituksena on löytää testattavasta ohjelmistosta virheitä. Virheitä halutaan löytää testaamalla, jotta virheet eivät haittaisi ohjelmiston todellisten käyttäjien ohjelmiston käyttöä. Testauksessa verrataan testattavan ohjelmiston käyttäytymistä toivottuun käyttäytymiseen. Testauksen tavoitteena voi myös ohjelmiston käyttäytymisen oikeellisuuden osoittaminen.

2.1 Testaamisen käsitteistöä

Ohjelmistoa, jonka toiminnallisuus halutaan varmistaa, kutsutaan testattavaksi ohjelmistoksi. Testitapaus on testattavan ohjelmiston yksittäinen testi, jolle on määritelty testisyötteet, alkutilanne, testin suorituksen vaiheet ja ehdot, joiden täyttäminen merkitsee testin onnistunutta suoritusta. Testisarja on sarja testitapauksia tai muita testisarjoja, jotka suoritetaan peräjälkeen tietyssä järjestyksessä.

Kun halutaan testata tiettyä toiminnallisuutta, testattava ohjelmisto pitää jollain konstilla saattaa tilaan, jossa toiminnallisuus on mahdollista testata. Ohjelmisto voidaan asettaa haluttuun tilaan joko ohjelmiston käyttöliittymän kautta ohjelmistoa operoimalla, luomalla haluttu tilanne ohjelmallisesti, tai manipuloimalla ohjelman tai sen osan riippuvuuksia. Riippuvuudet, niin ohjelmiston sisäiset kuin ulkoisetkin, voidaan testausta varten korvata testiä varten luoduilla toteutuksilla. Ohjelmistoa testattaessa tarvitaan erilaisia testisyötteitä. Erilaisilla syötteillä ohjelmiston toimintojen tulee mahdollisesti toimia eri tavoilla. Myös mahdolliset vialliset syötteet on hyvä testata ja varmistaa, että ohjelmisto käsittelee vialliset syötteet halutulla tavalla.

Testauksessa testattavaa ohjelmistoa testataan testitapauksen testidatalla. Automaatiotestauksessa testin suorituksessa automaatioskripti suorittaa testattavalla ohjelmistolla tehtävän. Testattava ohjelmisto asetetaan johonkin alkutilaan ja sille annetaan sarja testisyötteitä. Lopuksi verrataan testin lopputulosta, ohjelman tilaa testin jälkeen, testin odottamaan ohjelmiston tilaan.

Ennen kuin ohjelmistoa voidaan testata, tulee ohjelmiston testitapaukset suunnitella. Testitapausten suunnitteluun vaikuttaa testattava ohjelmisto, sekä testaukseen käytettävä testiohjelmisto. Testitapausten luonnissa, luodaan testaussuunnitelman mukaiset testitapaukset. Manuaalisten testien tapauksessa kirjoitetaan ohjeet testin maniaaliseen suorittamiseen. Automaattitestausta käytettäessä luodaan testiohjelmisto, johon kirjoitetaan testitapaukset sekä testattavan ohjelmiston automointi ohjelmakoodina. Testien suorittamisessa ohjelmistoa testataan testitapauksen mukaisesti ja sen toimintaa arvioidaan sekä verrataan odotettuun ohjelmiston toimintaan. Testitulokset voidaan tallentaa myöhempää vertailua ja arviointia varten. Testitapauksen suorittamisen ja tulosten arvioinnin jälkeen tuloksista laaditaan raportti, josta mahdollisen ohjelmiston virheet tai virheiden puute käy ilmi. Testauksen paljastamia ohjelmiston virheitä voidaan seurata ja selvittää pidemmän aikavälin trendejä tai samankaltaisten virheiden toistumista.

Testiautomaatiossa testijärjestelmää tulee ylläpitää, uusia testejä pitää luoda uusien toiminnallisuuksien testaamiseen, vanhoja päivittää toiminnallisuuksien muuttuessa ja tarpeettomia testejä pitää poistaa.

2.2 Testauksen tasot

Ohjelmistotestausta voidaan suorittaa eri tasoilla. Testaus tapahtuu eri ohjelmistokehityksen vaiheissa eri tavoilla. Lähimpänä ohjelmiston luontia, kun ohjelman osia kehitetään, tehdään yksikkötestit varmistamaan ohjelmiston osien toimivuus. Kun ohjelmiston osia liitetään yhteen, integroidaan, tehdään integraatiotestit. Ohjelmiston valmistumisen lähestyessä suoritetaan järjestelmätestaus ja käyttöönoton yhteydessä ohjelmistolle suoritetaan hyväksymistestaus. Perinteisessä ohjelmistotuotannossa nämä testausvaiheet vastaavat koko ohjelmiston elinkaarta ja sen työvaiheita. Iteratiivisissa ohjelmistotuotantomalleissa nämä testauksen tasot vastaavat paljon pienempien ja lyhyempien kehityssyklien työvaiheita.

Ohjelmistokehityksessä on neljä vaihetta. Nämä vaiheet ovat suunnittelu, toteutus, testaus ja ylläpito. Suunnitteluvaiheessa kartoitetaan tarpeet, jotka toteutettavan ohjelmiston tulee täyttää. Suunnitteluvaiheessa täsmennetään myös ohjelmiston arkkitehtuuri, eli ohjelmiston eri osat ja niiden väliset vuorovaikutukset. Toteutusvaiheessa luodaan itse ohjelmisto. Testausvaiheessa ohjelmiston ja sen yksittäisten osien toimintaa yksin tai toistensa kanssa vuorovaikutuksesa arvioidaan suunnitteluvaiheen määräämien vaatimusten perusteella. Kun ohjelmisto on ylläpidossa, sitä päivitetään ja jatkokehitetään muuttuvien tarpeiden tyydyttämiseksi tai vanhenevien teknologisten ratkaisujen päivittämiseksi.

Yksikkötestauksessa testataan koodiyksiköitä. Testattava yksikkö voi olla esimerkiksi funktio, luokka tai käännösyksikkö. Yksikkötestaukselle ominaista on testien suorittamisen nopeus, yksikön testaaminen itsenäisesti ja testattavien yksiköiden pieni laajuus. Yksikkötestauksessa testataan yksiköitä muusta toteutuksesta erillään niin, että yksiköiden riippuvuudet eivät vaikuta testattavan yksikön suoritukseen. Riippuvuudet korvataan testausta varten luoduilla toteutuksilla, mockeilla ja stubeilla. Stub, eli tynkä, on rajapinnan toteutus, joka vastaa täsmälleen niin kuin testin kirjoittaja kyseisessä tapauksessa haluaa [Fow07]. Mock on monimutkaisempi olio, joka testaa tilan lisäksi testattavan yksikön käyttäytymistä [Fow07].

Integraatiotestauksessa ohjelmiston eri osia testataan yhdessä. Integaraatiotestauksen esiehtona on, että yhdistettävät ohjelmiston osat on jo testattu yksikkötestitasolla. Integraatiotestauksen tarkoituksena on testata erillisten toimiviksi testattujen yksiköiden toimintaa yhdessä ja paljastaa virheitä näiden ohjelmiston osien välisistä rajapinnoista.

Järjestelmätestauksessa testataan koko ohjelmistoa ja sen kaikkien osien toimintaa yhdessä. Testitapaukset kulkevat koko järjestelmän läpi. Järjestelmätestauksessa testien suunnittelun pohjana on ohjelmiston määrittely.

Hyväksymistestauksessa testattavaa ohjelmistoa tutkitaan koko järjestelmän tasolla kuten järjestelmätestauksessakin. Erona järjestelmätestaukseen hyväksymistestauksessa testattavaa ohjelmistoa testataan todellisten käyttötarkoitusten ja toidellisten käyttäjien näkökulmasta, siinä missä järjestelmätestauksen lähtökohtana toimivat järjestelmän määrittelydokumenttit. Ideaalitilanteessa määrittelyt vastaavat todellisia käyttötapauksia, mutta näin ei kuitenkaan välttämättä ole. Ohjelmistojen rakentaminen voi olla hidasta ja kestää pitkiä aikoja. Tällöin käyttötapaukset voivat olla muuttuneet siitä mitä ne ovat olleet määrittelydokumentteja laadittaessa. Määrittelydokumenttien laatijat voivat myös olla eri ihmiset kuin ohjelmistot vastaanottajat ja lopulliset käyttäjät.

Hyväksymistestauksessa selvitetään kuinka hyvin testattava ohjelmisto toteuttaa tarkoitustaan ja kuinka valmis testattava ohjelmisto on käyttöönottoon.

Regressiotestaus on vanhan toiminnallisuuden jatkuvan hyväksyttävän toimimisen testaamista. Regressiotestauksen tarkoituksena on havaita testattavaan ohjelmistoon tehtyjen muutosten aiheuttamia virheitä [MEMON02]. Testattavaan ohjelmistoon tehtävien muutosten jälkeen suoritetaan regressiotestaussarja. Mikäli jokin sellainen testi, joka ennen muutoksia suoritettiin hyväksytysti, tuottaa muutosten jälkeen virheen, tiedetään muutosten rikkoneen jotain toiminnallisuutta joka aiemmin on toiminut.

Ohjelmistoa voidaan testata staattisesti, eli ohjelmakoodia tarkastelemalla, tai dynaamisesti, testattavaa ohjelmistoa suorittamalla. Staattisessa testauksessa tarkastellaan ohjelmakoodia ja pyritään löytämään virheitä sen rakenteesta. Staattisessa testauksessa testattavan ohjelmiston ohjelmakoodia ei lainkaan suoriteta [RT01]. Dynaamisessa testauksessa testattavaa ohjelmistoa suoritetaan keinotekoisissa testitilanteissa antamalla testattavalle ohjelmistolle testitapauksen kannalta kiinnostavia syötteitä ja tarkastellaan ohjelmiston suoritusta [RT01]. Suoritusta verrataan ennalta määriteltyihin odotuksiin [RT01].

Lasilaatikkotestauksessa testit rakennetaan testattavan ohjelmiston lähdekoodin perusteella. Lasilaatikkotestauksen tarkoituksena on testata jokainen testattavan ohjelmiston lähdekoodin osa. Lasilaatikkotestaus vaatii ymmärrystä ohjelmiston sisäisestä rakenteesta ja testaa ohjelmiston sisäistä toimintaa. Mustalaatikkotestauksessa tutkitaan toiminnallisuutta ilman tuntemusta ohjelmiston sisäisestä toteutuksesta tai sitä ei oteta huomioon. Testit luodaan ohjelmiston määrittelyn perusteella. Mustalaatikkotestaus soveltuu hyväksymistestaukseen, sillä hyväksymistestauksessa testataan ohjelmistoa käyttäjän näkökulmasta ja mustalaatikkotestauksessa testataan ilman ohjelmiston rakenteen tuntemusta välittäen vain ohjelmiston palauttamista vastauksista annettuun syötteeseen. Mustalaatikkotestauksessa testataan

toteuttaako ohjelmisto sitä tarkoitusta jota varten se on tehty. Harmaalaatikkotestauksessa testataan samalla tasolla kuin mustalaatikkotestauksessa tarkoituksena testata ohjelmiston toiminnallisuutta, mutta käyttäen ohjelmiston toteutuksen tuntemusta apuna testien suunnittelussa kuten lasilaatikkotestauksessa.

2.3 Automaattitestaus

Tässä aliluvussa selvitän mitä testiautomaatio on, miksi sitä tarvitaan ja miten ohjelmistojen testausta voidaan automatisoida. Testiautomaatio on minkä vain ohjelmiston testaamisen osa-alueen automatisointia.

Manuaalisessa testauksessa testaajana toimii ihminen. Testaaja käyttää testattavaa ohjelmistoa tarkoituksenaan selvittää toimiiko ohjelma siten kuin sen on tarkoitus toimia. Manuaalinen testaus on työlästä ja aikaavievää. Kun ohjelmistoa muutetaan, muutokset voivat aiheuttaa joko suoraan tai sivuvaikutuksenaan sen, että jokin vanha toiminnallisuus lakkaa toimimasta riittävällä tasolla.

Testiautomaatio on testien suorittamista ohjelmallisesti ilman, että ihmisen tarvitsee testata ja ohjata testattavaa ohjelmistoa manuaalisesti. Testiautomaation tehtävä on parantaa ohjelmiston laatua sekä vähentää työtä verrattuna manuaaliseen testaamiseen. Automatisoitujen testien suorittaminen ei vie yhtä paljon aikaa kuin testien manuaalinen suorittaminen ihmiseltä. Koska automatisoitujen testien suorittaminen ei vaadi testien suorittamiseen ihmistä, ei myöskään kulu kallista työaikaa. Automatisoidut testit voidaan tästä syystä suorittaa useammin kuin ihmisen suorittama testaaminen. Koska automatisoitujen testien suorittaminen on nopeampaa kuin manuaalisten testien, ja testit on mahdollista suorittaa useammin, on automaattitestien tärkein

tarkoitus olla kustannustehokkaampi tapa testata kuin manuaalinen testaaminen [BF12].

Ohjelmiston testauksesta voidaan automatisoida monia eri vaiheita. Tässä työssä keskitytään testattavan ohjelmiston hallinnan automatisointiin. Tällaisessa testauksessa automaatio-ohjelma tutkii testattavan ohjelmiston suorituksen aikaista tilaa ja syöttää sille komentoja sen tarjoamien rajapintojen, kuten käyttöliittymän, kautta. Tämän lisäksi on mahdollista automatisoida testien suorittaminen, esimerkiksi ajastaa testit suoritettavaksi joka päivä tiettyyn kellonaikaan tai aina kun tietty versionhallinnan haara päivittyy. Testitapausten luonnissa voidaan käyttää apuna automaatiota.

Testiautomaation käytön tuoman ajankäytöllisen hyödyn lisäksi yksi automaattisesti suoritettujen testien valtava etu ihmistestaajan tekemään testaukseen nähden on testaamisen tarkkuus [BR12]. Testiohjelmisto suorittaa testitapauksen jokaikinen testin suorituskerta samalla tavoin. Automatisoitu testi ei paina vääriä painikkeita, unohda testin vaiheita tai suorita testin vaiheita väärässä järjestyksessä, toisin kuin on mahdollista ihmistestaajan kanssa [BR12]. Visuaalisten käyttöliittymätestauksen testitapausten suunnitteleminen ja ohjelmoiminen sietämään ohjelmiston vikatilojen ja odottamattoman toiminnan sietämiseen on ensisijainen muuttuja testien luomiseen tarvittavassa työmäärässä [BR12].

Testiautomaation yleisiä virheitä on testien suorittaminen väärällä tasolla, liian optimistiset odotukset testien tuottamista säästöistä testaus- ja kehityskustannuksissa, sekä liian yksipuolisesti rakennettu testijärjestelmä, joka monesti testaa testattavaa ohjelmistoa lähinnä sen käyttöliittymän kautta [BWK05]. Automaatiota voidaan käyttää testauksen apuna myös muissa testauksen osa-alueissa kuin suorituksen automatisoinnissa, kuten testitapausten ja testiraporttien luonnissa [BWK05].

Testiautomaatio on sitä tehokkaampaa mitä useammin automaattitestit suoritetaan [BWK05]. Koska automatisoidun testin suorittaminen on nopeampaa kuin manuaalisen testin suorittaminen, suorituskertojen kasvaessa testistä saatu hyöty kasvaa. Automaattitestien laatu heikkenee nopeasti, jos niitä ei suoriteta usein [BWK05]. Testattavan ohjelmiston ohjelmakoodin muuttuessa sitä testaavat testit voivat lakata toimimasta oikein. Virheellisesti toimivat testit voivat raportoida testattavasta ohjelmistosta löytyvän virheitä, joita siitä ei todellisuudessa löydy. Jatkuva testien suorittaminen mahdollistaa testien hajoamisen aikaisen havaitsemisen ja korjaamisen. Testien pikainen korjaaminen ylläpitää luottamusta testeihin. Mikäli testit ovat pitkiä aikoja rikki, luottamus automaattitesteihin heikkenee jonka takia myös tahto testien suorittamiseen heikkenee. Testien suorittamisen harveneminen voi johtaa testattavan ohjelmiston virheiden havaitseminen myöhästymiseen lykäten samalla virheiden korjaamista ja heikentäen ohjelmiston laatua.

Automaattitestit eivät korvaa manuaalista testaamista [BWK05]. Suurin osa ohjelmiston virheistä löydetään testien luomisen aikana [BWK05]. Automatisoidut testit sen sijaan varmistavat, että testattava ohjelmisto toimii edelleen toivotulla tavalla. Automatisoidut testit vapauttavat ohjelmistotestaajat vanhojen testitapausten jatkuvalta uudelleen suorittamiselta. Testaajat voivat näin keskittyä löytämään testattavasta ohjelmistosta uusia virheitä.

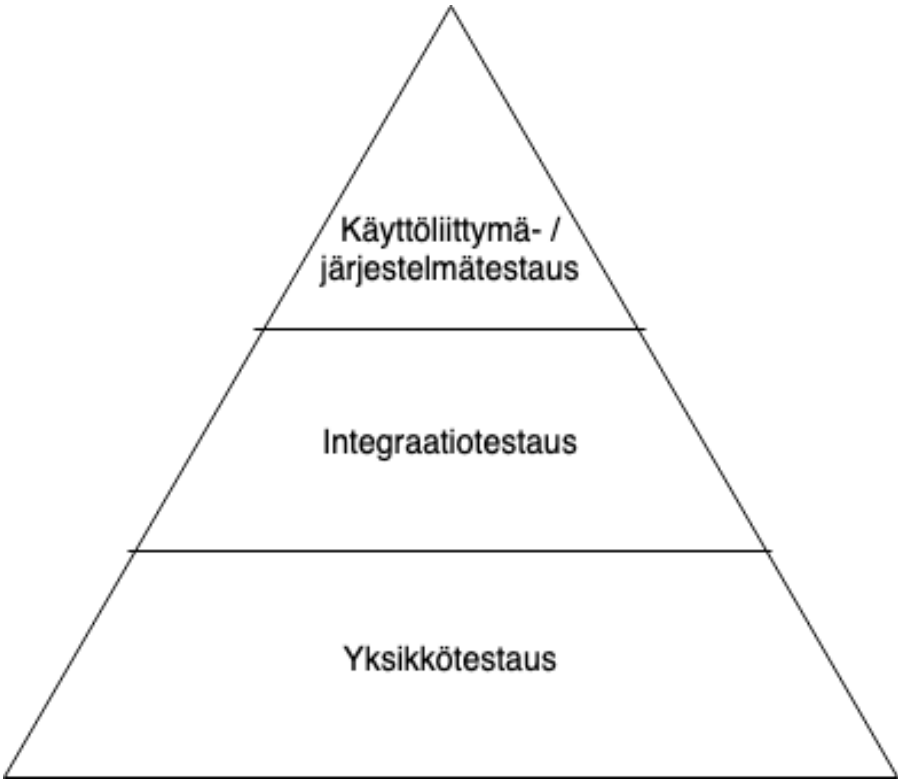
Testiautomaation ylläpito on vaikeaa [BWK05]. Kaiken ohjelmiston ylläpito on haastavaa, eivätkä automaattitestit ole tähän poikkeus. Testattavan ohjelmiston muuttuessa, osan kauttaaltaan testatun ohjelmiston testeistä on välttämättä muututtava myös.

Testiautomaatio mahdollistaa usein tehtävän regressiotestauksen. Useammin suoritetut testit havaitsevat ohjelmiston virheet nopeammin [SEL]. Testiautomaation avulla sovelluskehittäjät saavat nopeasti palautetta ohjelmistoon tehdyistä muutoksista [SEL]. Automatisoituja testejä voidaan

suorittaa mielivaltaisen usein [SEL]. Yleinen tapa on pystyttää testejä suorittava taustajärjestelmä, joka suorittaa ohjelmiston testit tai niiden osajoukon aina kun ohjelmiston ohjelmakoodille tehdään muutoksia. Tällaista järjestelmää ja prosessia kutsutaan jatkuvaksi integraatioksi, c*ontinuous integration.* Jatkuvan integraation avulla testit tulevat suoritetuiksi usein ja mahdolliset regressiot voidaan havaita, kirjata ja mahdollisesti korjata nopeasti. Jatkuvaa integraatiota varten voidaan varata oma suoritusympäristö, vaikka oma fyysinen tietokone tai suoritusajo voidaan tehdä ympäristö joka suorituskertaa varten pilveen automaattisesti pystytettävässä ympäristössä. Hyötynä on ohjelmistokehittäjien työkoneiden vapauttaminen aikaa vievien testiajojen suorittamiselta.

Kun testejä vodiaan suoritettaa automaattisesti, aukeaa mahdollisuus testata sama toiminnallisuus nopeasti useilla eri syötteillä ja odotetuilla tuloksilla. Datavetoisessa testaamisessa testien syötteet ja testin suorittava ohjelmakoodi erotetaan toisistaan niin, että testikoodi lukee suoritettaessa testitapausten syötteet ja odotetut tulokset erillisestä lähteestä ja suorittaa testit ja tulosten vertailun näillä luetuilla arvoilla. Lähde voi olla esimerkiksi tekstitiedosto tai taulukko. Tärkeää datalähteelle on uusien arvojen vaivaton syöttäminen ja lukeminen.

Testitapaukset toimivat eräänlaisena dokumentaationa sovelluksen toiminnasta. Ohjelmiston dokumentaatio on ohjelmiston toiminnan ymmärtämistä tukevaa tai selvittävää tekstiä ja kuvistusta. Testitapauksista voi nähdä miten ohjelmiston odotetaan toimivan ja testit suorittamalla selviää toimiiko sovellus todella odotusten mukaisesti. Varsinkin luonnollista kieltä muistuttavia DSL-kieliä käyttävät testityökalut soveltuvat luettavuutensa takia toimittamaan myös dokumentaation virkaan. DSL-kielet (*domain-spesific language*) ovat sovellusaluekohtaisia ohjelmointikieliä, jotka on luotu helpottamaan jonkin sovellusalueen ohjelmien kirjoittamista.



Kuva 1: Testaamisen pyramidimalli

Testipyramidi on malli, joka perustuu ajatukselle, että yksikkötestit ovat halpoja tehdä ja muokata sekä nopeita suorittaa, käyttöliittymätestit ovat kalliita ja hitaita kehittää, ja järjestelmätestit ovat tältä väliltä [Fow12]. Halpuutensa ja nopeutensa takia yksikkötestien tulisi kattaa valtaosa ohjelmiston testeistä [Fow12]. Vastaavasti käyttöliittymätestejä tulisi tehdä vain tapauksissa, joissa nopeammat ja halvemmat tekniikat eivät kykene testaamaan testattavaa ominaisuutta halutulla kattavuudella [Fow12].

2.4 Käyttöliittymätestaus

Käyttöliittymä on se ohjelmiston osa, joka kommunikoi ohjelmiston tilaa ohjelmiston käyttäjälle ja ottaa vastaan käyttäjän ohjelmistolle antamia syötteitä. Yleisiä käyttöliittymätyyppejä ovat komentorivikäyttöliittymä ja graafinen käyttöliittymä. Komentorivikäyttöliittymäohjelmistoja käytetään tietokoneen komentorivin kautta. Komentoriville annetaan tekstimuotoisia komentoja ja ohjelmisto esittää komentojen vastaukset tekstinä komentoriville. Graafisissa käyttöliittymissä on tekstisyötteen lisäksi erilaisia visuaalisia elementtejä, joita manipuloimalla ohjelmiston käyttäjä voi tutkia ja muuttaa ohjelmiston tilaa.

Käyttöliittymätestauksessa testataan testattavan ohjelmiston käyttöliittymää. Käyttöliittymätestit soveltuvat hyvin järjestelmä- ja hyväksymistestaukseen. Koko testattavaa ohjelmistoa voidaan testata käyttäjän näkökulmasta ohjelmiston käyttöliittymän kautta. Käyttöliittymiä voi myös yksikkötestata. Käyttöliittymän yksikkötestaamisessa ohjelmiston sisäinen toiminta korvataan tynkätoteutuksella, siten että toteutus tukee kyseistä testitapausta.

Käyttöliittymien automaattitestauksessa automaattitestin pitää pystyä vuorovaikuttamaan testattavan käyttöliittymän kanssa. Käyttöliittymästä pitää kyetä löytämään testattavat käyttöliittymäelementit. Samoin käyttöliitymän syötettä vastaanottaville elementeille pitää voida antaa syötteitä. Käyttöliittymän elementtejä voidaan paikantaa monin eri tavoin riippuen käyttöliittymän toteutustavasta. Selaimessa elementtejä voi etsiä esimerkiksi elementin tunnisteen tai css:n avulla. Visuaalisessa testauksessa pitää verrata ennalta otettua kuvaa ruudulle piirrettyyn kuvaan. Selainohjelmistojen automaatiotekniikoita käsitellään tarkemmin luvussa 4.

3 Web-ohjemistot

Web-ohjelmistoksi kutsutaan internetin yli käytettäviä ohjelmistoja joiden käyttöliittymä suoritetaan web-selaimessa. Web-ohjelmistot perustuvat asiakas-palvelin -malliin. Asiakasohjelmisto, tässä tapauksessa web-selain, lähettää pyynnön palvelimelle. Palvelin käsittelee pyynnön ja lähettää asiakkaalle vastauksen. Web-ohjelmistojen tapauksessa pyyntö on yleensä http-kutsu ja vastaus http-vastaus. Vastauksen sisältönä voi olla lähes mitä vain: mediaa, tekstimuotoista tai binäärimuotoista dataa. Usein vastaus sisältää html-dokumentin. Html-dokumentin pohjalta selain rakentaa web-sivun. Html-dokumentti määrittelee myös muita web-sivun rakentamiseen tarvittavia tiedostoja, kuten kuvia, javascript-tiedostoja ja css-tiedostoja.

Tässä luvussa käsitellään web-ohjelmistojen toimintaa. Aliluvussa 3.1 esitellään web-selain ja web-palvelin, sekä verkkoliikenne näiden välillä. Aliluvussa 3.2 käsitellään verkkosivujen toteusteknologioita HTML:ää, CSS:ää ja Javascriptia. Alivuvussa 3.3 esitellään verkkosivujen käsittelyn rajapinta dokumentin oliomalli, DOM. Aliluvun 3.4 aiheena ovat rikkaat käyttöliittymät verkkosivuilla.

3.1 Web-selain, web-palvelin ja niiden välinen liikenne

Web-selain on ohjelmisto, jonka tehtävänä on hakea ja näyttää sisältöä internetistä. Web-selaimia on monia erilaisia. Suosituimmat selaimet ovat Googlen kehittämä Chrome, Mozillan kehittämä Firefox, Microsoftin selaimet Intenet Explorer sekä Edge, ja Applen Safari [NMS]. Vaikka verkkoselaimet ja internet-teknologiat ovat vahvasti standardoituja, eri selaimet voivat toteuttaa joitakin toiminnallisuuksia hieman eri tavoin. Tämä lisää testaustaakkaa, sillä usein on mahdotonta tietää mitä selainta, saatikka mitä selaimen versiota,

ohjelmiston käyttäjät käyttävät. Näin ollen on suotavaa testata testattavaa ohjelmistoa mahdollisimman monella eri selaimella.

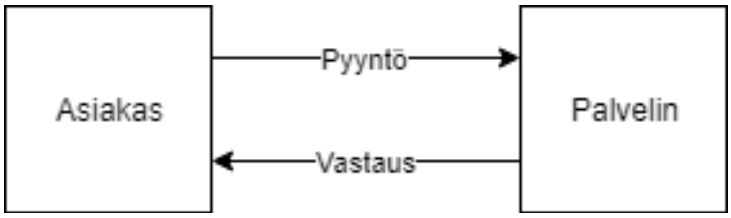
Päätön selain (*headless browser*) on verkkoselain, jolla ei ole graafista ulkoasua [Bid17]. Muuten ne toimivat kuin mikä tahansa muu selain. Sen sijaan, että selainta käytettäisiin kursorin ja näppäimistön avulla ja selain piirtäisi käyttöliittymän ikkunaan tilansa mukaisen graafisen kuvauksen, päätöntä selainta käytetään komentorivikäyttöliittymän tai mahdollisesti verkon yli tarjotun rajapinnan avulla.

Web-palvelin on ohjelmisto, joka tarjoaa sisältöä world wide webissä. Web-palvelin tarjoaa rajapinnan, jota voidaan kutsua verkon yli. Yleensä web-ohjelmistot perustuvat selaimessa suoritettavan käyttöliittymän ja web-palvelimen väliseen kommunikaatioon. Web-palvelimelta haetaan vähintään selaimessa suorittavan käyttöliittymän resurssit ja ohjemakoodi, mutta mahdollisesti käyttöliittymä tekee palvelimelle lisää kutsuja suorituksen aikana. Web-palvelin tarjoaa käyttöliittymälle erilaisia palveluja, kuten tiedon säilömistä ja hakemista tai laskentaa vaativia toimenpiteitä.

Web-ohjelmistot ovat hajautettuja asiakas-palvelin -mallin ohjelmistoja, jossa ohjelmiston tilaa ylläpidetään HTTP-kutsuilla ja -vastauksilla. HTTP-kutsut voivat olla asynkronisia eli ne suoritetaan käyttöliittymän taustalla ilman käyttäjän toiminnan keskeytymistä. Web-ohjelmistoja tehdään eri ohjelmointikielillä ja tekniikoilla selaimen päässä ja palvelimen päässä. Web-ohjelmistot ovat dynaamisia ja mahdollisesti epädeterministisiä [GMCM13]. Epädeterministmiä verkko-ohjelsmistoon tuo verkon yli tapahtuva liikenne [GMCM13]. Verkossa voi olla ruuhkaa, välityspalvelimia voi olla syystä tai toisesta saavuttamattomissa tai koko verkko voi olla verkkoselaimen tai web-palvelimen saavuttamattomissa.

HTTP on sovellustason tiedonsiirtoprotokolla, jota käytetään sisällön pyytämiseen ja tarjoilemiseen verkkoselainten ja web-palvelimien välillä [HTTP]. HTTP-kutsut muodostuvat URI:sta, HTTP-metodista, sekä kutsun otsakkeista ja kutsun sisällöstä [HTTP]. HTTP-sanomien siirrossa käytetään TCP-tiedonsiirtoprotokollaa, jonka avulla tietokoneet voivat ottaa yhteyttä toisiinsa ja toimittaa paketteja oikeassa järjestyksessä [HTTP]. Pyyntö-vastausmalli on ohjelmien välisen kommunikaation tapa.

Pyyntö-vastausmallissa yksi ohjelma lähettää toiselle ohjelmalle pyynnön jostakin resurssista. Pyynnön vastaanottava ohjelma pyynnön saadessaan käsittelee ja mahdollisesti suorittaa jonkin pyynnön mukaisen toiminnon. Lopulta pyynnön vastaanottanut ohjelma lähettää pyynnön lähettäneelle ohjelmalle jonkin vastauksen. Palvelin on ohjelmisto joka tarjoaa sisältöä tai toiminnallisuuksia muille järjestelmille. Toiminnallisuuksia käyttävät ohjelmistot ovat nimeltään asiakkaita. Ohjelmisto voi olla yhtä aikaa se asiakas, että palvelin.



Kuva 2: Asiakas-palvelin -malli

WebSocket on tapa selaimessa suoritettavien ohjelmistojen ja web-palvelimien väliseen kommunikointiin. WebSocket ei käytä web-ohjelmistojen yleisesti käyttämään HTTP-protokollaa viestien välitykseen. HTTP-protokollan tapaan WebSocket käyttää TCP-tiedosiirtoprokollaa. WebSocket tarjoaa web-ohjelmistojen käyttöön kaksisuuntaisen viestinvälityskanavan. Yksisuuntaisessa kanavassa vain yksi osapuoli lähettää viestejä, toinen osapuoli vain vastaa

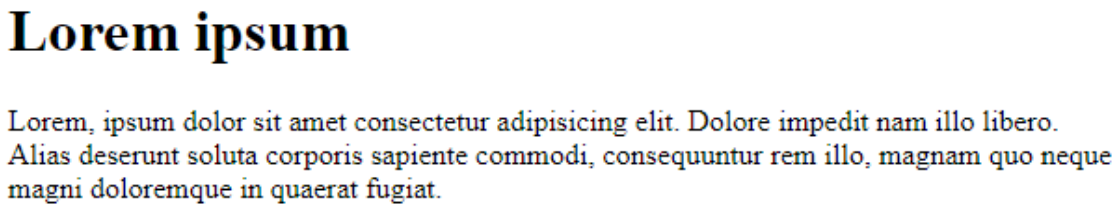
lähettäjä viesteihin. Kaksisuuntaisessa kanavassa molemmat osapuolet voivat lähettää viestejä. Web-palvelin voi siis lähettää selainohjelmistolle viestejä ilman selainohjelmiston pyyntöä.

3.2 HTML, CSS ja Javascript

Verkkosivut rakennetaan käyttäen HTML, CSS ja Javascript -teknologioita. HTML-dokumentti on HTML-kielellä kirjoitettu dokumentti. HTML on kieli, jolla kuvataan verkkosivujen rakennetta. HTML-dokumentin pohjalta verkkoselain rakentaa DOM-puun ja piirtää verkkosivun käyttäjän nähtäville. Staattinen sivu on dokumentti joka on aina samanlainen. Dynaaminen sivu on dokumentti joka voi muuttua ja joka voi olla eri pyyntökerroilla erilainen.

```
1  <html>
2      <head>
3          <title>Lorem ipsum</title>
4      </head>
5      <body style="width: 600px;">
6          <h1>Lorem ipsum</h1>
7          <p>Lorem, ipsum dolor sit amet consectetur adipisicing elit.
8              Dolore impedit nam illo libero. Alias deserunt soluta
9              corporis sapiente commodi, consequuntur rem illo, magnam quo neque magni
10             doloremque in quaerat fugiat.</p>
11     </body>
12 </html>
13
```

Kuva 3: Hyvin yksinkertainen sivu, jolla on muutama elementti.



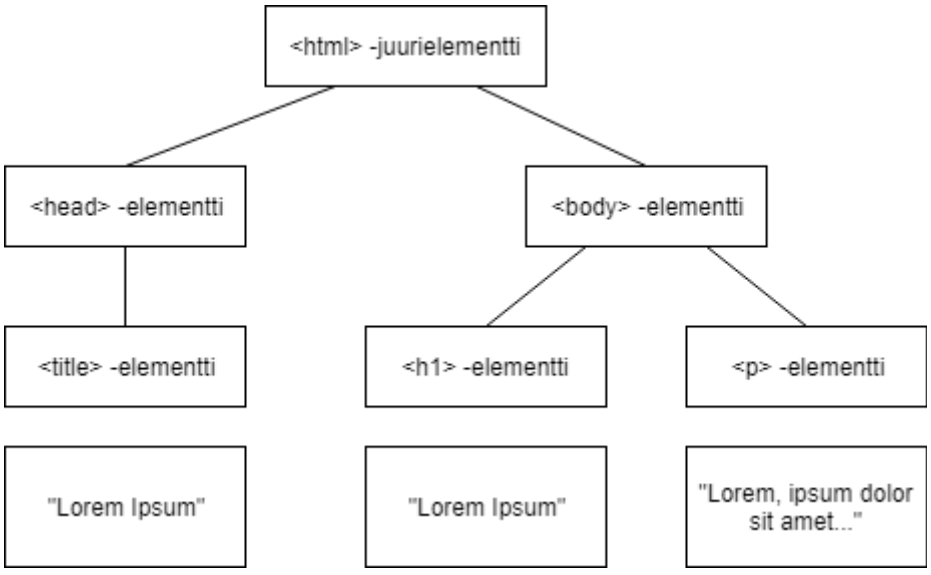
Kuva 4: Selaimen piirtämä sivu kuvan 3 HTML-koodista.

CSS, cascading style sheet eli ’laskeutuva tyylisivu’. CSS on kieli HTML-dokumenttien rakenteen ja ulkonäön muokkaamiseen. CSS on standardoitu W3C:n toimesta [CSS18]. CSS pitää sisällään säännöstön, jonka avulla valitaan HTML-elementtejä ja määritetään niiden tyyliominaisuuksia, kuten väriä, kokoa tai sijaintia verkkosivulla [CSS18]. CSS-valitsimia voidaan hyödyntää myös testaamisessa testattavien elementtien valitsemiseen [SLE]. CSS-valitsimilla voidaan paikantaa elementtejä niiden HTML-elementtityypin, tunnisteen, luokan, attribuutin ja pseudoluokan mukaan [SLE].

Javascript on ohjelmointikieli, jolle on suoritusympäristö web-selaimissa [MJS]. Javascript on alunperin luotu selaimia varten Brendan Eichin toimesta vuonna 1995 ja siitä tuli ECMA-standardi vuonna 1997 [W3JS]. Javascriptin avulla verkkosivuille voidaan lisätä erilaisia toimintoja. Javascript mahdollistaa modernit rikkaat web-sovellukset. Javascript on selaimien suorittama skriptikieli. Javascriptin avulla web-sivuista voidaan tehdä dynaamisia ja interaktiivisia. Javascriptilla voidaan muokata web-sivun sisältöä ja ulkonäköä, tehdä pyyntöjä web-palvelimille ja luoda uutta sisältöä. SSelaimien lisäksi Javascptiä voidaan suorittaa myös muissa ympäristöissä, kuten Node.js:llä. Node.js on Javascriptin suoritusympäristö, joka toimii Googlen Chromen V8 Javascript-moottorilla.

3.3 DOM

DOM on alusta- ja kieliriippumaton sovellusohjelmointirajapinta HTML- ja XML -dokmenttien käsittelyyn [MZDOM]. DOM on WHATWG:n standardoima [DOM]. DOM esittää dokumentin puutietorakenteena [DOM]. Dokumentin elementit ovat puun solmuja ja olioita [DOM]. DOM tarjoaa työkalut HTML-dokumenttien elementtien hakemiseen, muokkaamiseen, lisäämiseen ja poistamiseen [DOM]. Selaimeen ladatusta verkkosivusta luodaan puumallinen DOM, jonka juurisolmua kutsutaan dokumentiksi [DOM]. DOM-puun kaikki solmut ovat elementtejä. Elementillä voi olla attribuutteja, kuten nimi tai luokka. DOM-puun solmu voi olla myös solmulista (*nodeList*), joka on elementeistä koostuva lista [DOM]. Nimetty solmukartta (*namedNodeMap*) on lista solmuja, joita voidaan aksessoida nimellä [DOM].



Kuva 5: Kaavio kuvien 3 ja 4 HTML-sivusta havainnollistaa HTML-dokumentin ja sen oliomallin puumaisen rakenteen.

DOMin manipulointi on melko raskasta. DOMin virtualisointi on keino käsitellä DOMin sisältöä ilman että varsinaisesti käsitellään DOMia. DOMista luodaan erillinen malli, joka sisältää kaiken tiedon jonka DOM itsekin sisältää. Virtuaalisen DOMin käsittely on nopeampaa kuin itse oikean DOMin, sillä selaimen DOM-rajapintaa ei kutsuta ja mitään ei todellisuudessa piirretä ruudulle. Jsdom on javascript-kirjasto, joka totetuttaa WHATWG:n (web hypertext application technology working group) DOM- ja HTML- standardit [DOM, HTML] Node.js -ympäristössä [jsdom]. Jsdom pyrkii olemaan riittävän samanlainen selaintoteutusten kanssa, että sen avulla voidaan testata web-ohjelmistoja [jsdom]. Jsdom on siis täysin javascriptilla toteutettu DOM-implementaatio, joka toimii verkkoselaimen ulkopuolella. Jsdomia käyttäen web-ohjelmistojen käyttöliittymiä voidaan testata ilman, että tarvitsee käynnistää raskasta selainprosessia. Tällä tavoin saadaan lisää suoritusnopeutta testien suoritukseen. Verkkoselainten DOM-toteutusten eroavaisuuksien takia Jsdomia käytettäessä menetetään oikealla selaimella testattaessa saatu luottamus siihen, että testitapausten mukaiset toiminnot toimivat odotetusti juuri oikealla selaimella.

3.4 Ajax ja rikkaat käyttöliittymät

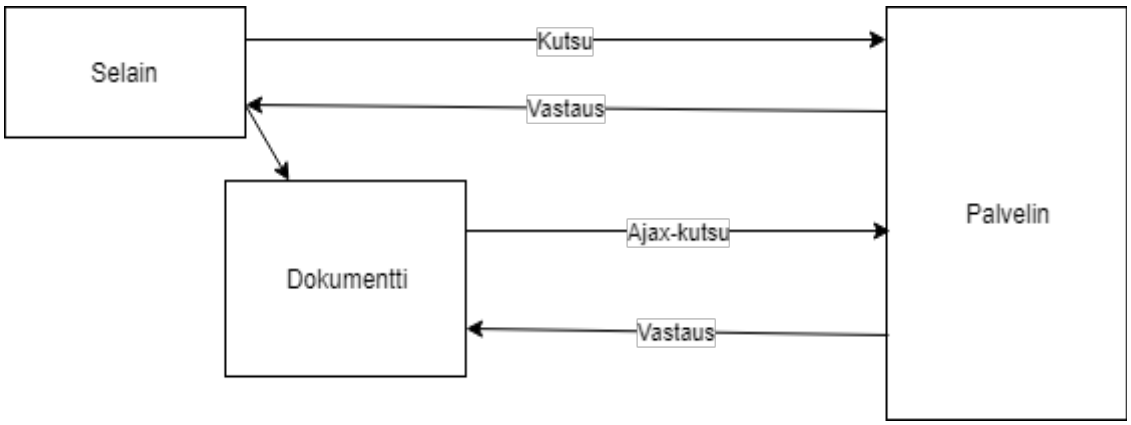
Perinteinen web-ohjelmiston malli, jossa ohjelmisto koostuu toisiin verkkosivuihin linkkejä sisältävistä verkkosivuista ja ohjelmiston tilasiirtymät tapahtuvat palvelimelle tehtävien HTTP-kutsujen avulla sisältää ohjelmiston käyttäjälle paljon odottelua. Sivulta toiselle siirtyminen vaatii HTTP-kutsun web-palvelimelle, palvelimen vastauksen odottamista ja koko sivun uudelleen piirtämistä. Ajax on tekniikka mahdollistaa selaimissa toimiville ohjelmille erittäin vuorovaikutteisen käyttöliittymän. Ajax-tekniikka on tapa yhdistää muutamia web-teknologioita ja rakentaa niiden avulla rikkaita web-ohjelmistoja.

Ajax on lyhenne sanoista asynkroninen Javascript ja XML [AJAX]. Teknologiat, jotka mahdollistavat Ajax-ohjelmistojen luomisen ovat HTML DOM sivun dynaamiseen muokkaamiseen, sekä XMLHttpRequest asynkroniseen tiedonsiirtoon dokumentin ja palvelimen välillä [AJAX]. Verkkosivu kommunikoi palvelimen kanssa jontain tekstiformaattia, esimerkiksi XML:ää tai JSON:a käyttäen [AJAX] . Selaimen suorittamalla JavaScriptilla kutsutaan selaimen DOM- ja XMLHttpRequest-rajapintoja ja sidotaan nämä yhteen [AJAX]. XML on harhaanjohtavaa, sanoma voi olla missä vain tekstimuotoisessa formaatissa, esimerkiksi JSON-muodossa. Selain tarjoaa XMLHttpRequest-oliona ohjelmointirajapinnan HTTP-kutsujen tekemiseen asynkronisesti [XHR]. Asynkroninen datan noutaminen mahdollistaa sujuvan web-sivuston käyttökokemuksen käyttäjälle. Kun tietoja haetaan asynkronisesti palvemilta, ei sivun käyttö keskeydy vaan käyttäjä voi jatkaa sivulla toimimista. DOM mahdollistaa sivun muokkaamisen dynaamisesti selaimessa javascriptin avulla. Asynkronisten HTTP-kutsujen noutamien tietojen perusteella sivua voidaan muokata vapaasti. Uudet näkymät voidaan rakentaa aukiolevaan sivuun ilman sivun uudelleenlatausta. Nimi XMLHttpRequest on historiallinen eikä enää täysin kuvaa toiminnallisuutta [XHR].

Single Page Application, yleisesti lyhennettynä SPA, on yhden verkkosivun tai dokumentin ohjelmisto. SPA-sivuilla sivun ulkonäköä ja sisältöä muokataan dynaamisesti sivun uudelleenlataamisen sijaan. Selaimen ladattua HTML-sivun, joka lataa javascript-tiedostot ja css-tiedostot. Javascript luo verkkosivun lisäämällä dokumenttiin elementtejä dynaamisesti hyödyntäen selaimen tarjoamaa DOM-rajapintaa. Javasciptin avulla noudetaan sisältöä verkkosivun taustapalveluista asynkronisesti käyttöliittymän taustalla ilman sivun uudelleenlatausta, sillä Javasciptin ja DOMin avulla voidaan päivittää vain niitä verkkosivun elementtejä, jotka on tarpeen päivittää.

Perinteisesten verkkosivujen tapauksessa selain lähettää web-palvelimelle HTTP-kutsun, johon web-palvelin vastaa lähettämällä paluusanomassa kutsun

vastauksena HTML-dokumentin. HTML-dokumentin sisältämien hyperlinkkien kautta käyttäjä voi käskeä selainta tekemään uusia HTTP-kutsuja web-palvelimelle, jolloin web-palvelin palauttaa uusia sivuja. Näin lähes jokainen käyttäjälle mahdollinen toiminto aiheuttaa synkronisen HTTP-sanomien vaihdon ja sivun täydellisen uudelleenlatauksen. Tänä aikana web-sivuston käyttäjä ei voi tehdä muuta kuin odottaa.



Kuva 6: Ajaxin avulla dokumentti voi tehdä palvelimelle pyyntöjä ilman sivun uudelleenlataamista.

Ajax-dokumenteissa koko dokumentin uudelleen lataaminen ei ole tarpeen. Javascriptilla voidaan tehdä HTTP-kutsuja web-palvelimelle ja muokata avoinna olevaa dokumenttia DOM-rajapinnan avulla. Ajax-sivujen yleisimmät ongelmatilanteet voidaa jakaa kahteen luokkaan [AASA]. Nämä luokat ovat virheellinen DOM-manipulaatio ja datan asynkronisesta siirrosta johtuvat virheet.

4 Web-ohjelmistojen testaaminen

Web-ohjelmistoilla on joitakin erityislaatuisia ominaisuuksia, jotka vaikuttavat niiden testaamiseen. Tällaisia ominaisuuksia ovat käyttöliittymän suorittaminen ulkopuolisessa ohjelmassa, web-selaimessa, sekä käyttöliittymän ja taustapalveluiden välinen internetin avulla tapahtuva kommunikaatio. Mitä moninmutkaisempi järjestelmän ympäristö, sitä vaativampaa ja vaivalloisempaa on järjestelmän testaaminen ja sen luotettavuuden varmistaminen [SNEED04]. Testaaminen on vaikeaa yksinkertaisessakin ympäristössä. Kun testataan web-ohjelmistoja, web-ympäristöstä tulee osa testausongelmaa [SNEED04]. Web-ohjelmistojen testauksella on kaksi erillistä tarkoitusta, ohjelmiston testaaminen sen ympäristössä ja itse ohjelmiston toiminnallisuuden testaaminen [SNEED04]. Ohjelmiston ympäristö ja arkkitehtuuri tulee ottaa huomioon web-ohjelmistoa testatessa [SNEED04]. Hajautetun ohjelmiston testaaminen maksaa kolme kertaa enemmän kuin perinteisten, yhdellä koneella suoritettavien, ohjemistojen testaus [DF06]. Asiakas-palvelin-ohjelmistojen testaaminen voi vaatia jopa puolet koko ohjelmiston kehitysprojektin työmäärästä [TR04]. Internetin liikeohjelmistojen monimutkaisuutta lisää se, että ohjelmisto voi käsittää useampia palvelimia [SNEED04].

Aliluvussa 4.1 käsittelen DOMin ja käyttöliittymän elementtien paikantimista ohjelmallisesti . Aliluku 4.2 käsittelee Javascriptia ja sen hyötyjä sekä ongelmia web-ohjelmiston automatisoinnissa. Aliluvun 4.3 aiheena on Webdriver. Aliluvussa 4.4 esitellään kuvantunnistus web-ohjelmiston automaation välineenä.

4.1 DOM ja paikantimet

DOM on alusta- ja kieliriippumaton sovellusohjelmointirajapinta HTML- ja XML -dokmenttien käsittelyyn. DOM esittää dokumentin puutietorakenteena. Dokumentin elementit ovat puun solmuja. DOM tarjoaa työkalut HTML-

dokumenttien elementtien hakemiseen, muokkaamiseen, lisäämiseen ja poistamiseen. Kun web-selain lataa HTML-dokumentin, se luo dokumentista oliomallin. Tämä oliomalli noudattaa puurakennetta, jonka solmuina toimivat dokumentin elementit. Puurakenteen ansiosta sisäkkäisiä elementtejä on nopeata etsiä suhteessa toisiinsa.

Paikantimet ovat yksi tärkeimpiä web-sovellusten testaamisen välineitä. Paikantimet mahdollistavat testattavan ohjelmiston käskyttämisen sen käyttöliittymän kautta ja sen elementtien tutkimisen ohjelmallisesti. Paikantimien avulla paikannetaan web-dokumenttien elementtejä ohjelmallisesti.

Web-sovelluksia testattaessa paikantimet ovat kyselyitä, joiden avulla HTML-sivulta haetaan HTML-elementtejä. Paikannettua elementtiä voidaan siten tarkastella tai siihen voidaan vaikuttaa. Paikantimen tulee olla mahdollisimman täsmällinen. Kyselyn tulisi löytää vain ne elementit, joista ollaan kiinnostuneita. Vääriä elementtejä löytävä paikannin aiheuttaa testien epäonnistumisia. Paikantimen toinen tärkeä ominaisuus tarkkuuden ohella on suorituskyky. Halutut elementit tulisi paikantaa mahdollisimman nopeasti. Testien nopea suoritusnopeus mahdollistaa testien frequest suorittamisen. Usein suoritetut testit havaitsevat testattavasta ohjelmistosta virheitä ja ongelmia nopeammin mahdollistaen virheiden ripeän korjaamisen. Myös paikantimien ymmärrettävyys, eli luonnin ja varsinkin muuttamisen tulisi olla mahdollisimman helppoa. Paikantimien vaivaton luominen ja muuttaminen nopeuttaa testien luomista ja kehittämistä vähentäen kehityksen kustannuksia.

Paikantimet voivat etsiä verkkosivulta elementtejä monin eri tavoin. Paikantimet voivat perustua HTML-dokumentin rakenteeseen, HTML-elementtien attribuutteihin, kuten id- tai name-attribuuttiin tai elementtien sisätöihin. Kuvantunnistukseen perustuva paikantaminen löytää jopa DOMin ulkopuolisia asioita, kuten canvasin tai jonkin embedded pluginin sisältöä. Paikantimelle tärkeää on tarkkuus. Paikantimen tulee löytää juuri oikea elementti. Paikannin,

joka ei löydä mitään tai väärän elementin, on täysin hyödytön testauksessa. Myös useamman elementin löytyminen on haitallista tilanteissa, joissa halutaan nimenomaan löytää jokin tietty dokumentin elementti. Paikantimen tulee olla myös mahdollisimman nopea. Testien suorituksen nopeus mahdollistaa testien suorittamisen usein. Paikantimen ymmärrettävyys vaikuttaa siihen, miten vaivatonta elementit tunnistavia paikantimia on luoda, sekä siihen, miten työlästä niiden myöhempi muuttaminen on. Paikantimet ovat web-testien herkin osa hajoamaan testattavan ohjelmiston muuttuessa [LSRT15].

4.2 Javascript

Javascript on ilmeinen valinta selaimessa suoritettavan ohjelmiston automatisoimiseen. Koska selain suorittaa Javascript-koodia ja tarjoaa monia rajapintoja verkkosivun tutkimiseen ja muokkaamiseen, on Javascript-pohjaisen automaation ja testaamisen aloittaminen helppoa. Lähes kaikista yleisistä verkkoselaimista löytyy kehitystyökalut, joilla pääsee nopeasti alkuun.

Kun avuksi otetaan testikirjastoja, kirjastojen javascript-koodi pitää saada upotettua mukaan testattavalle verkkosivulle. Yksi vaihtoehto on liittää testikirjastot mukaan testattavan sivuston lähdekoodiin. Tällöin testikirjastot latautuisivat käyttäjän selaimeen sivuston muiden resurssien mukana aina sivustoa käytettäessä ja olisivat kenen tahansa sivuston käyttäjän hyödynnettävissä. Suuret datamäärät ovat rikkaiden käyttöliittymien verkkosivujen yleinen vitsaus, ja ylimääräisten testausresurssien lisääminen käyttäjien verkkoliikenteeseen pahentaisi tilannetta. Vaihtoehtoisesti sivustoa voidaan kutsua välityspalvelimen (*proxy*) kautta testaustilanteessa. Välityspalvelin on palvelin, joka sijaitsee kutsuvan asiakkaan ja kutsuttavan palvelimen välissä, välittäen kutsun ja mahdollisesti muokaten välitettävää kutsua jollain tavalla. Välityspalvelimen avulla testattavaan sivustoon voidaan injektoida javascript-testikoodia, jonka avulla automaattitestit suoritetaan.

Välityspalvelinta käytettäessä vältytään testikoodin liittämistä testattavan sivuston lähdekoodiin.

Javascript-testaamisen yksi ongelma on yhden lähteen käytäntö (*single origin policy*) [SOP]. Origin on URL:n protokollan, ”hostin” ja portin määrittämä ominaisuus, jolla tarkoitetaan dokumentin alkuperää [SOP]. Saman alkuperän käytäntö on selaimen turvallisuuskäytäntö, jolla rajoitetaan dokumenttien ja scriptien pääsyä eri alkuperästä peräisin oleviin resursseihin [SOP]. Selaimessa sivujen Javascript-ympäristöt eristetään toisistaan turvallisuussyistä, jolloin verkkosivulla sijaitseva Javascript-koodi ei pääse käsiksi sivun ulkopuoliseen sisältöön, kuten toisessa välilehdessä sijaitsevaan verkkosivuun tai toisesta lähteestä peräisin olevan verkkosivun paikalliseen varastoon (*local storage*) [SOP]. Saman alkuperän käytäntö hankaloittaa pelkällä selaimessa suoritettavalla JavaScriptilla testaamista.

Cypress on javascript-pohjainen web-käyttöliittymien testausohjelmointikehys, jonka myyntivaltteja ovat asentamisen ja käytön aloittamisen vaivattomuus, sekä sille luotujen testien johdonmukaisuus ja varmuus [CYP]. Cypress-testit suoritetaan osittain testattavan ohjelmiston kanssa samassa javascript-ympäristössä web-ohjelmiston verkkosivulla ja osittain erillisessä Node.js prosessissa [CYP]. Koska cypress-testit suoritetaan selaimessa, javascript on ainoa tuettu ohjelmointikieli. Toisin kuin esimerkiksi Selenium, Cypress on täysi testauskehys, joka keskittyy End-to-end -testien luomiseen [CYP]. Cypress ei ole automaatiotyökalu, eikä pyri soveltumaan yleiseen web-sivujen automatisointiin.

Cypress-kehyksessä ei tule koskaan olemaan tukea useille selaimen välilehdille eikä se kykene hallitsemaan kahta selainta samanaikaisesti [CYP]. Cypress on tarkoitettu ohjelmistokehityksen tueksi, ohjelmiston testaamiseen ohjelmiston luomisen aikana [CYP]. Testattavan ohjelmiston kanssa samassa

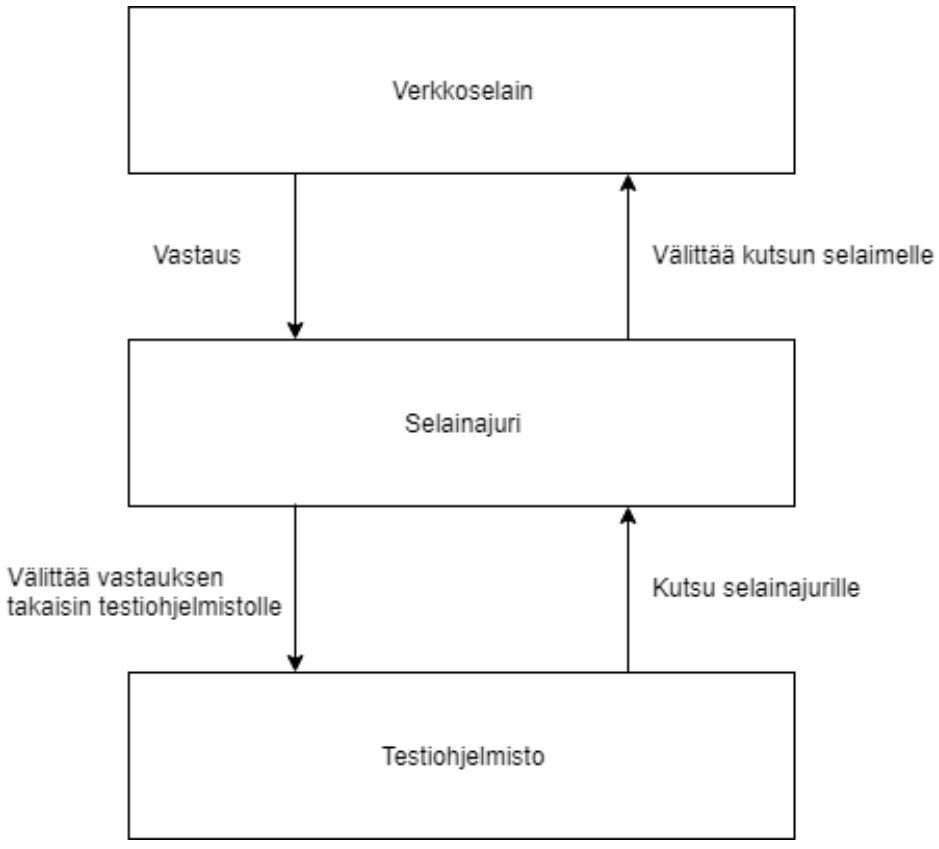
suoritusympäristössä suorittuminen mahdollistaa kaikkien sovelluksen tapahtumien havaitsemisen, niiden tutkimisen ja niihin reagoimisen.

4.3 Webdriver

WebDriver on etähallintaprotokolla, joka tarjoaa rajapinnan selaimen sisäisen tilan tarkkailuun ja hallintaan selaimen ulkopuolisesta prosessista [W3WD]. Rajapinnan kautta voidaan tutkia ja käsitellä web-dokumenttien elementtejä mm. DOM-rajapinnan avulla sekä hallita käyttäjäagentin toimintaa [W3WD]. Webdriver kutsuu selaimen omia automaatiotyökaluja [sel]. Eri selaimissa nämä työkalut on toteutettu eri tavoin ja kaikki vaativat oman selainajurinsa [sel].

Selenium WebDriver on vain automaatiokirjasto, jolla voidaan automoida verkkoselaimia. Selenium Webdriver ei ole täysiverinen testaustyökalu, vaan automaatiotyökalu joka soveltuu myös mainiosti testaamiseen. Moni verkko-ohjelmistojen testauskehys perustuu webdriver-protokollaan, jotkut käyttävät riippuvuutena Selenium Webdriveria, toiset toteuttavat protokollan täysin itsenäisesti. Webdriver.io on Node.js-testiohjelmointikehys, joka perustuu webdriver-protokollalle, jonka se itse toteuttaa [webdriver.io]. Protractor on testauskehys Angular-sovellusten testaamiseen. Protractor hyödyntää selaimen automoinnissa Selenium Webdriveria [protractor]. Nightwatch on Node.js:llä toimiva Webdriveriin perustuva end-to-end -testauskirjasto [nightwatch]. Appium on testauskehys, mobiililaitteille tehtyjen natiivien ja web-pohjaisten sovellusten testaamiseen [appium].

WebDriverin protokollassa kommunikoidaan kahdenlaisten ohjelmien välillä. Paikallinen pää (local end) toimii asiakasohjelmistona ja kähettää webdriver-komentoja etäpäälle (*remote end*) [W3WD]. Paikallinen pää on tavallisesti ohjelmointirajapinta webdriver-protokollan käyttöön. Protokollan palvelimena toimivat etäpäät. Etäpää voi olla välisolmu, joka välittää webdriver-komentoja eteenpäin kohti loppupistesolmua (*endpoint node*) [W3WD]. Välisolmut toteuttavat protokollan määrittelemät paikallisen pään että etäpään ominaisuudet. Selainajurit ovat välisolmuja. Loppupistesolmu on etäpää, joka toteuttaa webdriver-komentojen määrittelemien tehtävien, etäpään askelien (remote end steps), lopullisen suorittamisen [W3WD]. Selaimet ovat webdriver-protokollan loppupistesolmuja.



Kuva 7: Selainajuri toimii välityspalvelimena selaimen testiohjelmiston webdriver-kutsujen ja -vastausten välityksessä [W3WD]

Webdriver-protokolla koostuu HTTP-protokollan avulla toteutettavista komennoista [W3WD]. Komennot koostuvat URI-polusta ja HTTP-metodista. WebDriver-palvelin etäpää vastaanottaa HTTP-kutsuina toteutetut komennot ja ohjaa hallitsemaansa selainta komennon määrittämällä tavalla. Webdriver-pohjainen automaatio perustuu automaatioskriptin ja selainajurin väliseen Selenium-komentojen välitykseen. Selenium-komento on HTTP-kutsu, jonka automaatioskripti lähettää webdriver-asiakaskirjaston avulla selainajurin suorittamalle HTTP-palvelimelle. Selainajurin HTTP-palvelin ottaa vastaan kutsun, jonka jälkeen selainajuri kutsuu Selenium-komentokutsun pohjalta tarvittavia selaimen rajapintoja. Mahdollinen vastaus tai tulos palautetaan HTTP-vastauksen mukana takaisin selainajurilta itse automaatioskriptille.

WebDriverin rajapinta tarjoaa lukuisia selaimen toimintoja rajapintaa hyödyntäville ohjelmistoille. Webdriverin avulla käynnistetään selain, joka toimii selainajurinsa hallinnassa [W3WD]. Selain voidaan käskeä latamaan verkkosivu halutusta osoitteesta ja ladatulta sivulta voidaan etsiä UI-elementtejä lukuisia erilaisia paikantimia hyödyntäen [W3WD]. Paikannetuille elementeille voidaan välittää käyttäjän antamia syötteitä, ja painikkeille sekä muille vuorovaikutettaville elementeille voidaan välittää käskyjä [W3WD]. Webdriver voi komentaa selainta vaihtamaan välilehteä, sekä avata ja sulkea dialogeja ja tutkia niiden sisältöjä. Webdriver voi navigoida sivujen välillä, eikä ole sidottu yhteen lähteeseen [W3WD]. Webdriver pääsee myös käsiksi selaimen sivuhistoriatietoihin ja evästeisiin [W3WD]. Jotkin sovellukset muuttavat käytöstään riippuen selaimen käyttäjäagentista (*user agent).* Webdriver kykenee muuttamaan selaimen verkkosivulle kertoman käyttäjäagentin [W3WD]. Webdriver mahdollistaa latauslinkkien, dialog-ikkunoiden, ja muiden selainten tarjoamien toiminnallisuuksien testaamisen, joita monilla javascript-testaustyökaluilla ei voida testata [RSS17].

Selenium WebDriver tarjoaa seuraavat paikannusvaihtoehdot: elementin luokka, CSS, ID-attribuutti, linkin teksti ja osittainen linkin teksti, elementin nimi-attribuutti, elementin luokka-attribuutti, HTML-tunniste ja Xpath. XPath on syntaksi XML-dokumenttien elementtien määrittämiseen [XPATH]. XPathilla määritetään XML-dokumentista solmuja ilmaisemalla polku elementtiin [XPATH]. XPathia voi käyttää monilla eri ohjelmointikielillä. XPath -lauseet määrittävät polun haluttuun XML-elementtiin. Myös HTML-dokumenttien elementtejä voidaan määrittää XPathin avulla. Elementtejä voidaan tunnistaa helposti pelkän id-attribuutin avulla, mikäli sellainen vain on elementille annettu. Id-attribuutti on yksilöivä. Muun muassa nimi (*name*), luokka (*class*) ja id ovat HTML-tunnisteiden attribuutteja. Cascading style sheet eli ’laskeutuva tyylisivu’. CSS on tekniikka verkkosivujen rakenteen ja ulkonäön muokkaamiseen. CSS-tekniikassa paikannetaan verkkosivujen elementtejä omalla paikanninkielellä. Tätä paikannustapaa voidaan käyttää myös Seleniumissa. Id-paikantimet ovat parempia korjaamisen kannalta kuin Xpath-paikantimet [LCRS13]. Tämä johtuu id-paikantimen yksinkertaisuudesta verrattuna Xpath-paikantimeen. Id-paikannin tarkastelee vain yhtä tunnisteen attribuuttia, kun taas XPath-paikantimessa on paljon monimutkaisempi kysely. Linkin sisäisen tekstin ja id tai -Xpath-tunnisteiden yhteiskäyttö on vielä tarkempaa [LCRS13]. Web-sivujen kaikille elementeille on tehokkaampaa antaa id-attribuutit, kuin käyttää XPath-tunnisteita [LCRS13].

Sivuoliamallissa (*Page Object Pattern*) mallinnetaan testattava sivu oliona. Sivuoliomallin luonnissa käytetään samaa ohjelmoitikieltäkuin se, jolla testit kirjoitetaan. Sivuolio tarjoaa metodeinaan ja funkioinaan mallinnetun sivun toiminnot. Sivuolio kapsuloi testattavan sivun ominaisuudet ja toiminnallisuudet. Sivuolion metodeja voidaan kutsua testikoodista. Sivuolio toimii näin uudelleenkäytettävänä rajapintana testikoodin ja testattavan sivun välillä. Rajapinnan on tarkoitus piilottaa HTML-dokumentin kanssa käyty vuorovaikutus, sekä koota testisivun kanssa vuorovaikuttava ohjelmakoodi yhteen paikkaan. Kun testattavan sivun jokin osio muuttuu, sen aiheuttamat

muutokset kohdistuvat vain yhteen paikkaan eli kyseisen sivun oliomallin toteutukseen sen sijaan, että muutoksen vaikutuksia pitäisi metsästää pitkin testikoodia.

4.4 Kuvantunnistus

Tämä aliluku käsittelee kuvantunnistustekniikoiden käyttöä verkkosivujen testaamisessa. Selitän mihin kuvantunnistusta tarvitaan sekä miten se eroaa selaimen tarjoamiin rajapintoihin perustuvasta testaamisesta. Kuvantunnistustestauksessa verrataan ohjelmiston visuaalista tilaa ennalta otettuihin kuvankaappauksiin. Kuvankaappausten avulla paikannetaan käyttöliittymästä löytyviä elementtejä. Kun DOM-pohjaisessa automaatiossa elementtien paikantamisene käytetään tunnisteita, attribuutteja ja polkuja, kuvantunnistukseen perustuvassa automaatiossa tähän käytetään kuvia ja kuvien vertailua. Kuvantunnistukseen perustuvan testauksen on havaittu soveltuvan pitkäjänteiseen ohjelmistojen testaamiseen, vaikka sillä on muutama suuri ongelma [AF17]. Nämä ongelmat ovat kuvien hallinnan työläys ja mobiililaitteilla testaamisen puutteellisuus [AF17].

Kuvantunnistuksen avulla voidaan testata DOM-rajapinnan tarjoamien palveluiden ulkopuolisia asioita. Kuvantunnistuksen avulla voidaan testata koko selaimen ulkopuolisia asioita, kuten web-ohjelmiston tuottamien tiedostojen, esimerkiksi PDF-tiedostojen, sisältöä. Kuvantunnistuksen avulla on mahdollista testata sivuston elementtejä, jotka ovat DOMin näkökulmasta vain yksi elementti, mutta joka koostuukin todellisuudessa useasta loogisesta elementistä, joihin DOMilla ei ole pääsyä. Tällaisia voivat olla esimerkiksi ActiveX ja Flash -elementit [SLRT14]. Myös HTML canvas esiintyy DOMissa vain yhtenä elementtinä, jonka sisäistä tilaa ei pysty DOM-pohjaisilla työkaluilla selvittämään. Canvas-elementtiä käytettäessä piirretään suoraan monitorin

pikselipuskuriin täysin tilattomasti. Koska canvas ei muuta DOMia tai ylläpidä tilaa, canvas-pohjaisten elementtien testaaminen on haastavaa.

Kuvantunnistukseen perustuvat testit ovat riippumattomia testattavan ohjelmiston toteutusteknologiasta. Kuvatunnistukseen perustuvat testit ovat myös riippumattomia testattavan ohjelmiston suoritusalustasta [CYM10]. Kuvantunnistustyökalun ptiää kuitenkin olla saatavilla samalle käyttöjärjestelmälle tai alustalle kuin testattava ohjelmisto.

Kuvantunnistuksen käytöllä on tottakai myös haittoja. Kuvantunnistukseen perustuvien testien suorittaminen on raskasta ja vie enemmän aikaa kuin DOM-pohjaisten testien [LCRT14]. Testitapausten luonti on hitaampaa, samoin kuin testiprojektin ylläpito on työläämpää [LCRT14]. Kuvantunnistukseen perustuvat testit tuottavat herkemmin virheellisiä tuloksia testattavan ohjelmiston muutosten seurauksena [LCRT14].

Kuvantunnistukseen perustuvat testit vaativat kuvatiedostoja paikantimiksi ja testien vertailudataksi [LCRT14]. Testiprojektin kasvaessa kuvien määrä kasvaa, jolloin kuvien vaatima levytila kasvaa. Levytilaa suurempi ongelma on kuvatiedostojen versionhallinta. Binääritiedostojen versionhallinta on tekstitiedostojen versionhallintaa haastavampaa. Binäärimuotoisten artefaktien eroja ei voida samalla tavoin vertailla kuin tekstin. Versionhallinta on tärkeä osa ohjelmistojen kehitystä. Toimiva versionhallinta helpottaa testiohjelmiston ylläpitoa. Eri ohjelmiston versioiden välisten erojen vaivaton näkeminen auttaa ohjelmistoon päätyneiden virheiden korjaamista.

Sikuli on kuvankaappauksiin ja kuvantunnistukseen perustuva ohjelmistoautomaatiotyökalu [sikuli]. Kuvantunnistus on Sikulissa toteutettu OpenCV:llä [sikuli]. OpenCV on avoimen lähdekoodin kuvantunnistuskirjasto [OCV]. Kuvantunnistuksen lisäksi Sikulilla voidaan automatisoida hiiren kursorin liikkeitä ja hiiren painalluksia sekä näppäimistön näppäimien painalluksia

[sikuli]. Sikuli tukee myös tekstintunnistusta. Sikuli on Java-pohjainen ohjelmisto, joka toimii yleisimmillä käyttöjärjestelmillä: Windowsilla, Macilla ja Linuxilla [sikuli].

Koska Sikuli toimii kuvantunnistuksella, sillä voidaan automatisoida mitä vain tietokoneen ruudulle piirrettävää. Toisin kuin WebDriverilla ja JavaScriptilla, kuvantunnistukseen perustuvassa automaatiossa ei tarvitse olla mitään tuntemusta testattavan ohjelmiston toteutusteknologioista. Testattavan ohjelmiston ei tarvitse olla edes web-sovellus tai toimia selaimessa, kuvantunnistustestauksessa voidaan testata yhtä lailla natiiveja sovelluksia, eli sovelluksia jotka on toteutettu juuri käytettävälle alustalle. Kuvantunnistuksen soveltuvuus minkä tahansa graafisen käyttöliittymän omavaan ohjelmiston testaamiseen onkin sen suurin vahvuus.

Kuvantunnistusta voidaan automaation lisäksi käyttää visuaalisten regressioiden tunnistamiseen testattavasta ohjelmistosta. Visuaalisten regressioiden tunnistuksessa ohjelmistosta otetaan kuvakaappaus, jota verrataan oikeaksi tiedettyyn kuvaan samasta ohjelmiston tilanteesta [ATY18]. Kuvien vertailulla saadaan selville onko sivuston ulkonäkö muuttunut [ATY18].

5 Automaatiotekniikoiden arviointi

Tässä luvussa esitellään verkko-ohjelmistojen käyttöliittymien automaatiotekniikoita ja työkaluja. Eri tekniikat vuorovaikuttavat eri tavoin testattavan verkkosivun ja sen käyttöliittymäelementtien kanssa. Tekniikat eroavat myös siinä, miten automaatio-ohjelmaa suoritetaan ja siinä missä testiohjelmaa suoritetaan. Javascript-injektiossa testattavaan sivuun lisätään testauksen mahdollistava javascript-sovelluskoodi. Injektoitu javascript-koodi voi sisältää kaiken testaukseen vaaditun ja koko testisuoritus suoritetaan testattavan sivun javascript-ympäristössä. Toisissa tapauksissa injektoitu

javascript sisältää vain testattavan sivun kanssa vuorovaikutuksen mahdollistavaa koodia jota ohjataan erillisestä prosessista. WebDriverin kanssa koko selainta automoidaan erillisen selainkohtaisen selainajurin avulla. Kuvantunnistuksessa käyttöliittymäelementit tunnistetaan visuaalisesti kuvankaappauksiin vertailemalla ja testattavaa ohjelmistoa ohjataan automatisoimalla kursorin liikkeitä ja näppäimistön painalluksia käyttöjärjestelmätasolla.

Ensimmäisessä aliluvussa 5.1 esittelen ISO/IEC 9126 laatumallin ja sovellan sen laatimia laatuominaisuuksia testauskirjastojen arviointiin. Seuraavissa aliluvuissa käsittelen automaattitestaukselle tärkeitä ominaisuuksia käsiteltävien testauskirjastojen, Seleniumin, Sikulin ja Cypressin, kautta laatumallin ominaisuuksien avulla. Toisesssa aliluvussa 5.2 käsittelen eri tekniikoiden soveltuvuutta erilaisten verkko-ohjelmistojen testaamiseen. Aliluku 5.3 keskittyy testien luomisen ja ylläpidon työläyteen valituilla testaustekniikoilla. Aliluku 5.4 keskittyy testaustekniikoiden välisiin nopeuseroihin testien suorituksen aikana. Aliluvussa 5.5 käsittelen testikirjastojen alusta- ja selaintukea.

5.1 Testityökalujen ominaisuudet

Tässä aliluvussa käsitellään testityökalujen verrattavia ominaisuuksia. Ohjelmistojen laadun arvioinnin standardi ISO/IEC 9126 määrittelee laatumallin, jossa on kuusi laadullista pääominaisuutta. Nämä ominaisuudet jakaantuvat vielä aliominaisuuksiin. Kuusi pääominaisuutta ovat toiminnallisuus, luotettavuus, käytettävyys, tehokkuus, ylläpidettävyys ja siirrettävyys. Pääominaisuudet jakaantuvat vielä aliominaisuuksiin, jotka on listattu taulukossa 1.

toiminnallisuus	soveltuvuus

	tarkkuus
	yhteentoimivuus
	turvallisuus
luotettavuus	kypsyys
	virheensietokyky
	palautettavuus
käytettävyys	ymmärrettävyys
	opittavuus
	operoitavuus
tehokkuus	ajalliseen tehokkuus
	resurssitehokkuus
ylläpidettävyys	analysoitavuus
	muutettavuus
	vakaus
	testattavuus
siirrettävyys	mukautuvuus
	asennettavuus
	korvattavuus

Taulukko 1: ISO/IEC 9126 ominaisuudet

Automaatio tarjoaa mahdollisuuden suorittaa testejä milloin vain ilman ihmistestaajien aiheuttamia kustannuksia. Mikäli testien suoritus on riittävän nopeaa, voidaan testit suorittaa vaikka jokaisen ohjelmakoodiin tehdyn muutoksen jälkeen. Tällöin virheet saadaan nopeammin havaittua ja korjattua. Tässä mielessä testien suoritusnopeus on hyvin tärkeä testijärjestelmän ominaisuus. Toinen hyvin merkittävä automaatiotestauksen työkalun ominaisuuksia on sen soveltuvuus testattavan ohjelmiston testaamiseen. Soveltuvuudella tarkoitetaan tekniikan ominaisuuksien soveltuvuutta testattavan

ohjelmiston eri toimintojen testaamiseen. Esimerkiksi pdf-dokumentteja tuottamista ei voida testata DOM-parsintaan perustuvalla testityökalulla, koska pdf-dokumentti ei toteuta DOM-rajapintaa. Tähän tarkoitukseen sopii paremmin kuvantunnistusta käyttävä testaustyökalu.

Ylläpidettävyys on projektin vaatimien muutosten toteuttamisen vaivattomuutta. Automaattitestauksessa tämä tarkoittaa vanhojen testitapausten muokkaamisen ja korjaamisen aiheuttamaa vaivaa tai sen vähyyttä. Korkean ylläpidettävyyden testiprojektin testejä on helppo analysoida ja testeihin on helppoa tehdä muutoksia. Käytettävyys automaattitestien luonnissa, kehityksessä ja käytössä on testikoodin sekä testiraporttien ymmärrettävyyttä, testien luomisen ja testikirjastojen ominaisuuksien opittavuutta. Luotettavuudella tarkoitetaan kuinka luotettavasti testi suorittaa testitapauksen, eli kuinka suurella todennäköisyydellä testi löytää virheen virheellisestä ohjelmasta tai kuinka varmasti testi hyväksyy oikein toimivan ohjelman. Ihmistestaajaan verrattuna automaattinen testi on yleisesti varmempi, sillä automaattitesti suorittaa testin joka kerta samalla tavalla, eikä unohda testin vaiheita tai suorita epähuomiossa vääriä toimintoja. Siirrettävyyden verkkosovellusten testaamisen kontekstissa ymmärrän tarkoittavan mahdollisuutta suorittaa testit eri verkkoselaimia vasten. Siirrettävyydellä voidaan tarkoittaa myös mahdollisuutta suorittaa testit eri käyttöjärjestelmiä käyttäen.

5.2 Soveltuvuus

Testausmenetelmien ja -teknologioiden tulee soveltua testattavan ohjelmiston testaamiseen. Verkko-ohjelmistojen tapauksessa testausmenetelmien tulee soveltua verkkoselaimessa ja verkkopalvelimilla suoritettavien ohjelmistojen automatisointiin ja tutkimiseen. Verkko-ohjelmistojen testaamisessa tulee ottaa huomioon verkko-ohjelmistoille ominaiset ongelma- ja virhetilanteet.

Automaatiotyökalujen yleiskäytettävuus ja soveltuvuus erilaisten käyttöliittymien automatisointiin ja testaamiseen on hyödyllistä [MEMON02].

Selaimessa suoritettavan käyttöliittymän elementtien paikantaminen ja paikannettujen elementtien ohjelmallinen käsittely ovat verkko-ohjelmistojen automaattitestauksen perusta, jota iltan käyttöliittymän kautta tehtävää automaattitestausta ei voi mielekkäästi tehdä. Sekä Cypress, että Selenium Webdriver käyttävät elementtien paikantamisessa ja käsittelyssä DOM-rajapintaa. Sikulilla tehtävissä testeissä elementtien paikantaminen ja käsittely tapahtuu kuvavertailun avulla. Aivan kaikkea verkkosivulla olevaa sisältöä ei voi DOM-rajapinnan avulla käsitellä, kuten ActiveX-, Flash- tai canvas-elementtejä. Mikäli näiden elementtien testaaminen halutaan osaksi verkko-ohjelmiston käyttöliittymävetoisia järjestelmätestejä, tarvitaan jokin muu automaatiotekniikka DOM-rajapinnan tilalta tai lisäksi. Sikuli ei käytä DOM-rajapintaa käyttöliittymän automatisoinnissa, vaan sen kuvantunnistukseen perustuva toiminta soveltuu mainiosti DOM-rajapinnan ulottumattomissa olevien elementtien testaamiseen.

Rikkaat Javascript-pohjaiset verkko-ohjelmistot, kuten yhden sivun ohjelmistot tai muut Ajaxia käyttävät ratkaisut ovat hyvin muodikkaita 2010-luvulla. Niille ominaista on asynkronisesti suoritettava verkkoliikenne käyttäen XMLHttpRequest-rajapintaa tai WebSocketteja, ja sivun dynaaminen muokkaaminen ilman verkkosivun uudelleen lataamista ja piirtämistä. Rikkaiden javascript-ohjelmistojen tilan ja tilasiirtymien hallinta suoritetaan dokumentin sisältämän Javascript-koodin avulla HTTP-kutsuille pyydettävien uusien sivujen noutamisen sijaan.

Verkkosivuun injektoitava Javascript-pohjainen testi, joka suoritetaan samassa selaimen Javascript-ympäristössä kuin itse verkkosivun sisältämä Javascript koodi pystyy pureutumaan tällaiseen ohjelmistoon erittäin syvälle. Cypress-testit injektoidaan testattavaan verkkosivuun [CYP]. Sen avulla voi kuunnella kaikkia ja reagoida kaikkiin tapahtumiin verkkosivulla. Cypressin dokumentaatio

rohkaiseekin testien kirjoittajia käyttämään hyväkseen Cypressin suomaa testattavan verkkosivun täydellistä hallintaa käyttämään oikoteitä ja muokkaamaan verkkosivun tilaa, elementtejä ja sivun verkkoliikennettä palvelemaan testitapausten vaatimuksia [www.cypress.io/how-it-works]. Cypress suosii näin lasilaatikkotestausta. Cypressin mukaan testien ei ole tarpeellista testata kaikkea verkkosivujen toimintaa juuri kuiten käyttäjä ja muokata ohjelmiston tilaa vain käyttöliittymän tarjoamien mahdollisuuksien mukaan, vaan testien tulee hyödyntää Cypressin tarjoamaa mahdollisuutta päästä käsiksi verkkosivulla suoritettavan ohjelman tapahtumiin [www.cypress.io/how-it-works].

Javascript soveltuu hyvin rikkaiden Javacript-pohjaisten webohjelmistojen testaamiseen. Kuvantunnistukseen perustuva testaus soveltuu parhaiten sellaisten käyttöliittymien testaamiseen, jotka sisältävät DOM-rajapinnan ulottumattomissa olevia elementtejä. Webdriver soveltuu tilanteisiin, jossa käyttöliittymän elementit on paikannettavissa DOMin avulla ja testauksessa tarvitaan ominaisuuksia, joita Javascript-injektioon perustuvassa testauksessa ei ole, kuten useampien selaimen välilehtien hyödyntäminen.

5.3 Testien luominen ja ylläpito

Testien luominen on testien ylläpidon ohella suurin testaamiseen menevä kulu, sillä testien luominen ja ylläpito vievät aikaa ja työtä. Eri elementtien paikannustekniikat vievät eri määrän aikaa. Kuvantunnistukseen perustuvien testien luominen vie enemmän aikaa kuin DOM-pohjaisten testien [LCRT14].

Datavetoinen testaaminen on helmpompaa DOM-pohjaisessa testauksessa kuin visuaalisessa testauksessa [LCRT14]. Visuaalisessa testauksessa tulee olla kuvankaappaukset olemassa testien assertioiden validointiin [LCRT14].

DOM-pohjaisessa testauksessa on helppoa erottaa web-elementin lokaattori ja sisältö toisistaan [LCRT14]. Visuaalisessa tesktauksessa lokaattori ja testattava sisältö ovat yksi ja sama asia [LCRT14]. Visuaalisessa testauksessa ei siten ole mahdollista tehdä täysin datavetoista testijärjestelmää [LCRT14].

Testien automatisointi tuottaa testiprojektin, joka vaatii ylläpitoa, kuten mikä tahansa muu ohjelmistoprojekti.Testattavan ohjelmiston muutokset aiheuttavat muutoksia myös testeissä ja testattavan ohjelmiston jatkokehitys vaatii uusien testien luomista. Kun testattava ohjelmisto muuttuu ja testijärjestelmän paikantimet vaativat päivittämistä, muutosten vaatima työmäärä on tärkeä testijärjeslmän ominaisuus. Ohjelmiston muutokset voivat aiheuttaa testien nk. hajoamista, eli vanhat testit eivät enää suoriudu muuttuneen ohjelmiston testaamisesta johtuen toiminnallisuuden muutoksista tai käyttöliittymän elementtien muutoksista.

Kun testattava ohjelmisto muuttuu, testijärjestelmän käyttämät paikantimet voivat lakata toimimasta oikein. DOM-paikantimet ovat pääsääntöisesti lujatekoisempia kuin kuvantunnistuksella elementtien etsiminen [LCRT14]. Joissain tapauksissa kuvantunnistukseen perustuvat paikantimet sietävät paremmin testattavan ohjelmiston muutoksia [LCRT14]. Kuvantunnistuksessa käytettävien referenssikuvien ylläpito on kallista [AF17]. Visuaalisen testin rikkinäisen visuaalisen paikantimen löytäminen ja selvittäminen on helpompaa kuin esimerkiksi DOM-testauksen Xpath-paikantimen [LCRT14]. Kuvantunnistukseen perustuvien paikantimien päivittäminen kuitenkin vaatii yleisesti enemmän työtä kuin DOM-paikantimien käytö [LCRT14].

Selenium Webdriver suosii sivuolimallin käyttöä testien laatimisessa [Stew15]. Sivuoliomallin käyttö voi merkittävästi vähentää testattavan ohjelmiston muutoksista aiheutuvien testiohjelmiston vaatimien muutosten viemää aikaa,

sekä muutettavien koodirivien määrää [LCRS13]. Yleisiä testiohjelmiston muutoksia ovat DOM-elementtien paikanninlauseiden ja assertioiden muuttaminen [CSMR14]. Paikanninlauseiden sijainti on sivuoliomalleissa keskitetty sivuolioihin. Paikanninlauseiden keskittäminen ja uudelleenkäyttö vähentää näiden usein muuttuvien ohjelmakoodiosioiden muuttamisesta aiheutuvaa työmäärää.

DOM-pohjaiset testit ovat kestävämpiä kuin kuvantunnistukseen perustuvat testit, eli ne sietävät paremmin testattavaan ohjelmistoon tehtäviä muutoksia [LCRT14]. DOM-pohjaiset testit ovat myös nopeampia luoda sekä korjata, ja siten halvempia kustannuksiltaan kuin kuvantunnistukseen perustuvat testit [LCRT14]. Kuvantunnistukseen perustuvien ja DOM-pohjaisten automaattitestien paikantimien määrä saman toiminnallisuuden testaamiseen eroaa merkittävästi toisistaan. Kuvantunnustuspaikantimia vaaditaan enemmän noin 45% enemmän [LCRT14]. Paikantimien määrä vaikuttaa testien luomiseen, ylläpitoon ja korjaamiseen vaadittuun aikaan. Suuremmessa määrässä paikantimia löytyy todennäköisemmin toimimasta lakanneita paikantimia testattavan ohjelmiston muuttuessa.

DOM-paikantimiin perustuvat testit ovat nopeampia tuottaa ja ne sietävät paremmin testattavan ohjelmiston muutoksia kuin kuvantunnistukseen perustuvat testit. Sen lisäksi, että Webdriver testejä voidaan kirjoittaa käytännössä millä tahansa ohjelmointikielellä, erilaisia Javascript- ja Webdriver -testikirjastoja on liian lukuisia, että niiden ylläpidettävyyden eroavuuksista on mahdotonta sanoa mitään.

5.4 Testien suorittamisen tehokkuus

Automaattitestien suorittamiseen kuluva aika voi olla tärkeä testijärjestelmän ominaisuus. Vaikka automaattitestit suoriutuvat testien suorittamisesta

merkittävästi ihmistestaajaa nopeammin, on eri automaattitestauksen tekniikoiden välisen testien suoritusnopeuden vertailu silti mielekästä. Testien suorittamisen nopeus vaikuttaa siihen, miten usein ja missä tilanteissa testit on mahdollista suorittaa. Kattava manuaalinen testaus voi kestää päiviä tai viikkoja. Kattavaa manuaalista testausta ei siten voi tehdä joka ikisen ohjelmiston koodimuutoksen jälkeen. Monet seikat vaikuttavat automaattitestien suoritusnopeuteen. Käytetyt paikantimet ja tekniikka, jolla selaimen automaation suoritetaan, sekä testattavan verkko-ohjelmiston ja testiohjelmiston välinen kommunikaatio ja siihen valittu tekniikka ovat kaikki suuressa roolissa vaikuttamassa siihen kuinka nopeasti automaattitestit suoriutuvat.

Testiohjelmiston ja testattavan ohjelmiston välinen kommunikaatio vaikuttaa suuresti testien suoritusnopeuteen. Webdriveria käytettäessä jokaisen verkkosivun automaation komennon täytyy kulkea testiohjelmistolta selainajurin kautta HTTP-viestinä selaimelle [W3WD]. Cypress-testikoodi suoritetaan suurimmaksi osaksi selaimessa verkkosivun javascript-ympäristössä, joka on prosessien välistä kommunikaatiota nopeampaa [Tay19]. Tämän ansiosta Cypress-testit ovat yleisesti nopeampia suorittaa kuin Webdriver testit [Tay19]. Kunnollisia tutkimuksia Cypressin ja Webdriver-testien suoritusnopeuksien välillä ei ole tehty, mikä olisikin oiva tutkimuskohde.

Myös testauksen taso vaikuttaa testijärjestelmän suorituksen kestoon. Korkean tason järjestelmä- ja käyttöliittymän kautta testattavaa ohjelmaa automatisoivat testit ovat hitaampia kuin ohjelmiston komponenttien yhteistoimintaa ja komponenttien välisiä rajapintoja testaavat integraatiotestit, jotka puolestaan ovat hitaampia kuin yksittäisiä ohjelmiston komponetteja testaavat yksikkötestit. Näin ollen testikirjaston soveltuvuus ja keskittyminen tietyn tasoisiin testeihin vaikuttaa siihen millaisia testejä testikirjaston avulla luodaan, ja kuinka nopeasti nämä testit suoriutuvat.

Verkko-ohjelmiston testaaminen käyttöliittymän kautta vaatii ympäristön, johon ohjelmiston verkkosivut voidaan ladata, piirtää ja suorittaa. Yleisesti tämä ympäristö on verkkoselain. Kokonaisen verkkoselaimen käyttäminen ohjelmiston suoritusalustana on raskasta. Ratkaisuna tähän toimivat päättömän selaimet (*headless browser*). Sekä Selenium Webdriver, että Cypress tukevat päättömiä selaimia. Koska päätön selain ei piirrä graafista käyttöliittymään, se on huomattavasti graafisen käyttöliittymän omaavaa selainta nopeampi. Sikuli ei tue päättömiä selaimia, sillä sen automaatio perustuu kuvantunnistukseen.

Paikantimien käyttämä aika paikantimien löytämiseen on yksi testien suoritusnopeuteen vaikuttava osa. DOM-paikantimia käyttävät testit vievät vähemmän aikaa kuin kuvantunnistukseen perustuvia paikantimia käyttävät testit [LCRT14]. Webdriveria käyttävät ja selaimeen testattavaan sivuun injektoitavaan Javacriptiin perustuvat testikirjastot käyttävät pääosin DOMia käyttäviä paikantimia. Vaikka kuvantunnistukseen perustuva testaus on hitaampaa kuin Webdriveriin tai injektoituun Javascriptiin perustuja DOM-pohjainen testau, on kuvantunnistukseen perustuva testaus on silti merkittävästi nopeampaa kuin ihmistestaajan tekemä manuaalinen testaus. Erään Saabilla tehdyn tutkimuksen mukaan kuvantunnistukseen perustuvalla tekniikalla suoritettava testiajo oli noin 78% prosenttia nopeampaa kuin saman testiajon suorittaminen kokeneen ohjelmistotestaajan toimesta [AF17].

Testien suorittamisen tehokkuus ja nopeus on tärkeä ominaisuus, sillä se mahdollistaa testien suorittamisen usein. Cypress-testit suoritetaan suurimmaksi osaksi selaimessa verkkosivun omassa javasciptin suoritusympäristössä ja ovat näin ollen erittäin nopeita suorittaa. Selenium Webdriverissa komennot kulkevat HTTP-kutsuina selainajurin kautta testiohjelmiston ja selaimen välillä, tehden siitä Cypress-testejä hitaamman. Kuvantunnistukseen luottavat testit ovat joukosta kaikkein hitaimpia.

5.5 Alusta- ja selaintuki

Eri selaimet ja niiden eri ohjelmistoversiot eroavat toiminnallisuuksien ja verkon standardien toteutuksissa [MCBR]. Eroista johtuvien haittojen ja ongelmien välttämiseksi ohjelmistot tulee testata useammilla eri selaimilla. Tästä syystä testien automaation tuki useille eri selaimille ja alustoille on tärkeä ominaisuus.

Javascriptia voidaan suorittaa millä vain selaimella, sillä kaikki selaimet tukevat javascriptin suorittamista. Eri selaimet saattavat tukea Javascriptin eri versioita, jolloin jotkin testien käyttämät tuoreempien Javascriptin versioiden tarjoamat toiminnalisuudet eivät välttämättä ole tuettuja juuri sillä selaimella ja selaimen versiolla, jolla halutaan testata. Jotkin Javascript-pohjaiset testikirjastot toimivat vain joillain selaimilla.

Webdriver vaatii jokaiselle testeissä käytettävälle selaimelle oman selainajurin. Jokaista selainta varten tarvitaan siten oma suoritettava ajuritiedosto, samoin jokaista testien suoritusympäristöä kohden. Selenium Webdriverilla on tuki viidelle suurimmalle selaimelle: Googlen Chromelle, Microsoftin Internet Explorerille ja Edgelle, Mozillan Firefoxille, sekä Applen Safarille [SEL]. Webdriveria voidaan käyttää monella eri ohjelmointikielellä. Ohjelmointikielituki löytyy ainakin JavaScriptille, Javalle, C#:lle, Pythonille, Rubylle, PHP:lle, ja Perlille [SEL]. Laajasta ohjelmistokielten kirjosta on hyötyä, sillä näin useampi kehittäjä ja testaaja pystyy käyttämään Webdriveria itselleen tai projektiinsa sopivammalla ohjelmointikielellä. Cypress tukee pelkästään Chromemaisia selaimia, Chromea, Canarya, Chromiumia ja Electronia [CYPB].

Sikulin tapauksessa eri selainten välisistä eroista suurin ongelma on selaimen ulkonäkö, kuten ponnahdusikkunoiden ulkonäkö ja mahdollisesti eri selaimien käyttämät erilaiset fontit. Testien siirtäminen toiselle koneelle ja suorittaminen toisella koneella voi aiheuttaa ongelmia kuvantunnistukseen pohjautuvassa

testauksessa. Tietokoneissa voi olla eri näyttökoot tai selaimen perusfontti voi erilainen ja eri kokoinen [LCRT14]. Joillain kuvantunnistukseen perustuvilla testityökaluilla ei voida helposti testata sovellusten toimintaa mobiililaitteilla [AF17]. Erilaisia puhelinmalleja, joissa on testattavan sovelluksen ulkonäköön vaikuttavia seikkoja, kuten erikokoisia näyttöjä ja käyttöjärjestelmäversioita on valtavasti.

	Cypress	Webdriver	Sikuli
Chrome	x	x	x
Firefox		x	x
Edge		x	x
Internet Explorer		x	x
Safari		x	x

Taulukko 2: Automaatiokirjastojen toiminta eri selaimissa.

Selenium Webriver tukee kaikkia suosituimpia selaimia. Cypressilla taas voidaa testata vain chrome-pohjaisia selaimia. Sikulilla voi testata kaikkia mahdollisia selaimia, sillä se vuorovaikuta suoraan selaimen kanssa vaan selainta suorittavan käyttöjärjestelmän kanssa. Sikulilla tosin saattaa joutua luomaan eri selaimille eri referenssikuvat, johtuen selaimien eroista.

6 Yhteenveto

Tässä työssä on esitelty modernien verkko-ohjelmistojen toimintaperiaatteet ja mahdollistavat teknologiat, ohjelmistotestauksen ja testiautomaation peruskäsitteistöä ja metodiikkaa, sekä käsitelty selaimessa suoritettavien internetin yli hajautettujen ohjelmistojen käyttöliittymävetoisen testaamisen

haasteita ja testaamisen työkaluja ja menetelmiä. Olen vertaillut eri menetelmiä toisiinsa, keskittyen testiautomaation lupauksia täyttäviin ominaisuuksiin. Testiautomaation tarkoituksena on nopeuttaa testaamista, mahdollistaen laajempien testiajojen suortittamisen mahdollisimman usein. Tämä siirtää testaamisen työmäärää ihmisen suorittamasta testaamisesta automaattitestien suunnitteluun ja toteuttamiseen sekä automaattitestiprojektin ylläpitoon. Näin ollen erilaisten selainkäyttöliittymävetoisten automaatio- ja testausmenetelmien tärkeimmät verrattavat ominaisuudet ovat automaattitestien laatimisen nopeus, automaattitestien suorituksen nopeus, sekä testiprojektin vaatiman ylläpidon määrä ja nopeus. Lisäksi ensiarvoisen tärkeää on verrata eri menetelmien soveltuvuutta erilaisten web-sovellusten automatisointiin ja testaamiseen.

Visuaalisen testaaminen ei vaadi tuntemusta testattavan ohjelmiston toteutuksen yksityiskohdista. Testitapausten luomiseen riittää ymmärrys siitä miten ohjelman tulisi käyttäjän näkökulmasta toimia. Koska testit perustuvat vain ruudulle piirrettyyn visuaaliseen dataan, mahdollistaa kuvantunnistukseen perustuva testaus sellaisten käyttöliittymien tai niiden elementtien testaamisen, joihin ei voida vaikuttaa selaimen tarjoamien rajapintojen kautta. Samoilla testaustyökaluilla voidaan myös luoda testitapauksia muille kuin ohjelmistoille kuin web-ohjelmistoille.

Visuaalisen testaamisen suurimmat ongelmat ovat testien laatimisen hitaus, testien ylläpidon hitaus, testien muihin ratkaisuihin nähden hidas suoritusnopeus, sekä herkkyys virheisiin.

Webdriver on yleisesti käytetty automaatiotekniikka web-testauksessa. Webdriverin ongelmat ovat hyvin tunnettuja, joten testejä luodessa ja muokatessa ratkaisuja ja apua mahdollisiin ongelmiin on helpompaa löytää kuin jonkin vähemmän käytetyn ratkaisun ongelmiin. Webdriver välittää testien komennot itse testattavaa sivua suorittavalle selaimelle, eikä suoraan testattavalle sivulle. Kun automaatiokerros ohjaa selainta sivun sijaan, vältytään

yhden lähteen käytäntöön ja yhteen selaimeen tai selaimen välilehteen juuttumisen aiheuttamilta ongelmilta.

Webdriver-protokollassa selainta ja testattavaa ohjelmistoa ohjataan selaimen ulkopuolisesta prosessista http-protokollan avulla. Sanomien serialisointi ja deseralisionti JSON-muotoisen ja käsittelevän ohjelman sisäisen oliomuodon välillä, sekä itse sanomien lähettäminen tuovat testiensuoritukseen huomattavasti lisää suoritusaikaa lisäävää työtä.

Javascriptiin perustuvan automaation suurimmat edut ovat testien laatimisen aloittamisen helppous ja matala aloituskynnys, sekä suoritusnopeus. Varsinkin testityökalu, jota suoritetaan samassa ympäristössä eli testattavan verkkosivun javascript-hiekkalaatokossa kuin testattavan ohjelmiston käyttöliittymä, on paljon kuvantunnistukseen tai webdriveriin perustuvia testausmenetelmiä nopeampaa. Web-sovellusten luonnissa tarvitaan lähes väistämättä javascriptia, joten web-sovellusten kehittäjät osaavat sitä, mikä edelleen madaltaa kynnystä javascript-pohjaisen testaustyökalun käyttöönottoon.

Mikäli javascript-testauskirjaston testikoodi suoritetaan kokonaisuudessaan testattavan ohjelmiston verkkosivulla, sitoo testikoodia selaimen yhden alkuperän menetelmä. Tällöin testityökalu ei kykene siirtymään toisen lähteen alla olevalle verkkosivulle. Testityökalu ei voi myöskään hallita useampaa selainta yhtäaikaisesti eikä avata useampaa välilehteä.

Tässä työssä on vastattu tutkismuskysymyksiin millaisia erilaisia tekniikoita web-ohjelmistojen automaatiotestaamiseen on mielekästä käyttää, ja kuinka ne eroavat toisistaan. Erilaiset menetelmät soveltuvat erilaisiin tilanteisiin. Javascript-pohjaiset testaustyökalut ovat kaikkein nopeimpia, mutta selaimessa verkko-ohjelmiston kanssa samassa suoritusympäristössä toimiminen asettaa testijärjestelmälle joitakin rajoitteita. Webdriverin avulla näistä osa voidaan kiertää, mutta samalla joudutaan uhraamaan testien suoritusnopeutta.

Kuvantunnistuksen avulla voidaan testata minkälaisia graafisen käyttöliittymän omaavia sovelluksia tahansa pelkkien web-sovellusten sijaan, sekä sellaisia verkkosivuun upotettuja elementtejä, joihin ei voida DOM-rajapinnan avulla päästä käsiksi. Näistä menetelmistä kuvantunnistus on kaikkein monikäyttöisin.

Lähteet

AASA Ali Mesbah, Arie Van Deursen, An Architectural Style for Ajax. 2007 *Working IEEE/IFIP Conference on Software Architecture*

AF17 Emil Alegroth, Robert Feldt. On the long-term use of visual gui testing in industrial practice: a case study. *Empirical Software Engineering* Volume 22 Issue 6, December 2017 Pages 2937-2971

AJAX Jesse James Garrett, Ajax: A new approach to web applications, 2005

BF12 Emil Borjesson, Robert Feldt, Automated System Testing Using Visual GUI Testing Tools: A Comparative Study in Industry. *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation,* 17.4.2012

Bid17 Eric Bidelman, Getting Started with Headless Chrome, 2017, https://developers.google.com/web/updates/2017/04/headless-chrome , [13.10.2019]

BWK05 S. Berner, R. Weber, R.K. Keller, Observations and lessons learned from automated testing. *Proceedings. 27th International Conference on Software Engineering*, 2005.

CDTP Chromen DevTools -Github-sivu, https://chromedevtools.github.io/devtools-protocol/, [1.8.2019]

CSRM14 Laurent Christophe, Reinout Stevens, Coen De Roover,

Wolfgang De Meuter, Prevalence and Maintenance of Automated Functional Tests for Web Applications. *2014 IEEE International Conference on Software Maintenance and Evolution,* 12.2014.

CSS18 CSS Snapshot 2018, https://www.w3.org/TR/css-2018/, [12.10.2019]

CYM10 Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller, GUI testing using computer vision. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '10). ACM, New York, NY, USA, 1535-1544, 10.4.2010.

CYPB Cypress.io -kotisivu, Launching Browsers, docs.cypress.io/guides/core-concepts/launching-browsers.html

CYPR Cypress.io -kotisivu, Limitations, https://docs.cypress.io/guides/guides/web-security.html#Limitations

DF06 G. A. Di Lucca and A. R. Fasolino, Testing Web-based applications: The state of the art and future trends. *Information and Software Technology*, vol. 48, no. 12, pp. 1172–1186, 2006.

DOM DOM Living Standard, dom.spec.whatwg.org/, [16.10.2019]

Fow07 Martin Fowler, Mocks Aren't Stubs, 2007. https://martinfowler.com/articles/mocksArentStubs.html [2.7.2019]

Fow12 Martin Fowler, Test Pyramid.

https://martinfowler.com/bliki/TestPyramid.html, 1.5.2012. [1.12.2017]

GMCM13 V. Garousi, A. Mesbah, A. Betin-Can, and S. Mirshokraie, A systematic mapping study of web application testing. *Information and Software Technology*, vol. 55, no. 8, pp. 1396–1374, 8.2013.

HTML HTML Living Standard, https://html.spec.whatwg.org/#goals , [9.8.2019]

HTTP Hypertext Transfer Protocol – HTTP/1.1, https://tools.ietf.org/html/rfc2616, [11.10.2019]

JSDOM Jsdom github-sivu, https://github.com/jsdom/jsdom, [1.8.2019]

LCRS13 M. Leotta, D. Clerissi, F. Ricca, and C. Spadaro. Improving test suites maintainability with the page object pattern: an industrial case study. *Proceedings of the 6th International Conference on Software Testing, Verification and Validation Workshops*, ICSTW 2013, pages 108-113. IEEE, 2013.

LCRT13 Maurizio Leotta, Diego Clerissi, Filippo Ricca, Paolo Tonella Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. 2013 20th *Working Conference on Reverse Engineering* (WCRE), sivut 272-281

LCRT14 Maurizio Leotta, Diego Clerissi, Filippo Ricca, Paolo Tonella Visual vs. DOM-Based Web Locators: An Empirical Study. *International Conference on Web Engineering* ICWE 2014: Web Engineering sivut 322-340

LSRT15 Maurizio Leotta, Andrea Stocco, Filippo Ricca, Paolo Tonella, Using Multi-Locators to Increase the Robustness of Web Test

Cases. 2015 IEEE 8th *International Conference on Software Testing, Verification and Validation* (ICST)

MEMON02 A.M. Memon, GUI testing: pitfalls and process, *Computer* (Volume: 35 , Issue: 8 , Aug 2002) Page(s): 87 – 88

MJS JavaScript, https://developer.mozilla.org/en-US/docs/Web/JavaScript , [13.10.2019]

MZDOM Introduction to the DOM, https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction , [15.10.2019]

nightwatch Nightwatch.js-kotisivu. https://nightwatchjs.org/

NMS Browser Market Share, https://www.netmarketshare.com/browser-market-share.aspx, [12.10.2019]

NODE Node.js-kotisivu, https://nodejs.org/en/

OCV OpenCV -kotisi, vuhttps://opencv.org/, [24.7.2018]

PR86 Paul Rook, Controlling software projects. 1986 IEEE *Software Engineering Journal* (Volume: 1, Issue: 1), sivut 7-16

protractor Protractor.js -kotisivu. https://www.protractortest.org/

SEL01 Seleniumin kotisivu, Intorduction. https://www.seleniumhq.org/docs/01_introducing_selenium.jsp , [4.12.2017]

SLE Baiju Muthukadan, Locating Elements, https://selenium-python.readthedocs.io/locating-elements.html , [13.10.2019]

SLRT14 Andrea Stocco, Maurizio Leotta, Filippo Ricca, Paolo Tonella, PESTO: A Tool for Migrating DOM-Based to Visual Web Tests. *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation,* 28.9.2014

SLS14 Andreas Spillner, Tilo Linz, Hans Schaefer, Software testing fundamentals.

Sneed04 H. M. Sneed, Testing a Web application. *6th IEEE International Workshop on Web Site Evolution*, 2004, pp. 3–10.

SOP Same Origin Policy, https://www.w3.org/Security/wiki/Same_Origin_Policy, [20.4.2018]

Stew15 Simon Stewart, Page Objects, 2015, https://github.com/SeleniumHQ/selenium/wiki/PageObjects , [25.10.2019]

Tay19 Gil Taylor, Cypress vs Selenium WebDriver: Better, or just different? 2019, https://applitools.com/blog/cypress-vs-selenium-webdriver-better-or-just-different , [12.10.2019]

W3JS W3Schools Javascript Tutorial, www.w3schools.com/js/, [2.6.2018]

W3WD WebDriver, W3C Living Document 19 September 2019. https://w3c.github.io/webdriver/, [1.10.2019]

webdriver.io Webdirver.io-kotisivu. https://webdriver.io/, [1.7.2019]

XHR XMLHttpRequest Living Standard, https://xhr.spec.whatwg.org, [10.10.2019]

Zai19 Vitali Zaidman, An Overview of JavaScript Testing in 2019, 18.2.2019. https://medium.com/welldone-software/an-overview-of-javascript-testing-in-2019-264e19514d0a, [20.8.2019]